Testability analysis and test-point insertion in RTL VHDL specifications for scan-based BIST

Article *in* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems · October 1999

DOI: 10.1109/43.784124 · Source: IEEE Xplore

4 authors, including:

Bozena Kaminska
Simon Fraser University
378 PUBLICATIONS **4,371** CITATIONS

SEE PROFILE

Testability Analysis and Test-Point Insertion in RTL VHDL Specifications for Scan-Based BIST

Samir Boubezari, *Member, IEEE,* Eduard Cerny, *Senior Member, IEEE,* Bozena Kaminska, *Member, IEEE,* and Benoit Nadeau-Dostie, *Senior Member, IEEE*

***Abstract*—This paper proposes a new testability analysis and test-point insertion method at the register transfer level (RTL), assuming a full scan and a pseudorandom built-in self-test design environment. The method is based on analyzing the RTL synchronous specification in synthesizable very high speed integrated circuit hardware descriptive language (VHDL). A VHDL intermediate form representation is first obtained from the VHDL specification and then converted to a directed acyclic graph (DAG) that represents all data dependencies and flow of control in the VHDL specification. Testability measures (TM's) are computed on this graph. The considered TM's are controllability and observability for each bit of each signal/variable that is declared or may be implied in the VHDL specification. Internal signals of functional modules (FM's) such as adders and comparators are also analyzed to compute their controllability and observability values. The internal signals are obtained by decomposing at the RTL large FM's into smaller ones. The calculation of TM's is carried out at a functional level rather than the gate level, to reduce or eliminate errors introduced by ignoring reconvergent fanouts in the gate network, and to reduce the complexity of the DAG construction. Based on the controllability/observability values, test-point insertion is performed to improve the testability for each bit of each signal/variable. This insertion is carried out in the original VHDL specification and thus becomes a part of it unlike in other existing methods. This allows full application of RTL synthesis optimization on both the functional and the test logic concurrently within the designer constraints such as area and delay. A number of benchmark circuits were used to show the applicability and the effectiveness of our method in terms of the resulting testability, area, and delay.**

***Index Terms*—Built-in self-test, register transfer level, testability analysis, testability measures, test point insertion.**

I. Introduction

NOWADAYS, register transfer level (RTL) synthesis or logic synthesis has become an integral part of a design process of digital circuits. Most industrial digital designs use automated RTL synthesis and we can thus achieve design-for-testability (DFT) by incorporating test and synthesis into a single methodology that is as automated as possible. Indeed, considering testability during design synthesis can reduce the overall design and manufacturing time. Even more important, the testability enhancement at the entry level to a synthesis tool makes it independent of the tool and the implementation technology. It becomes part of the design specification and may be optimized with the other synthesis tasks in terms of area and delay.

The main objective of our method is thus to raise the level of abstraction at which testability analysis and test-point insertion are performed. We propose a new testability analysis and test-point insertion method at the RTL, assuming full scan and pseudorandom built-in self-test (BIST) design environment. Full scan in combination with pseudorandom patterns is widely adopted in the industry due to its ease of implementation and fault diagnostic. Unfortunately, the presence of random pattern resistant faults in many practical circuits poses a serious limitation to its success. The solutions to tackle this limitation can be broadly classified as those that modify the input patterns or those that modify the circuit-under-test. In this paper, we are interested in the second class of solutions, circuit modifications, that introduce test points to improve the random pattern testability of a circuit. Our goal is to analyze and modify the very high speed integrated circuit hardware descriptive language (VHDL) RTL description of the circuit, in order to generate an easily testable gate-level circuit by a pseudorandom sequence under the BIST environment. This is the main advantage and motivation of this work. That is, whatever the complexity of the circuit, our objective is to apply synthesis compilation and optimization technology directly to a testable VHDL description, thus optimizing functional and inserted test logic concurrently, rather than introducing testability after the VHDL has been compiled to gate-level.

The proposed method uses as the starting point a VHDL specification given at the synthesizable synchronous RTL. It is analyzed to produce an intermediate representation, called the VHDL intermediate format (VIF), and transformed into a DAG on which testability analysis is performed by computing and propagating testability measures (TM's) forward and backward through the VHDL statements. The TM's are the controllability and the observability for each bit of each signal/variable. Internal signals of functional modules (FM's) such as adders, comparators, and multiplexers are also analyzed to determine their controllability and observability values. The internal signals are obtained by decomposing at the RTL large FM's into smaller FM blocks, each such

Manuscript received November 19, 1997; revised November 17, 1998. This work was supported by LogicVision (Canada) and Micronet Centre of Excellence (Canada). This paper was recommended by Associate Editor J. Rajski.

S. Boubezari was with the Electrical Engineering Department, École Polytechnique de Montréal, PQ, Canada H3C 3A7. He is now with Synopsys, Inc., Mountain View, CA 94043-4033 USA (e-mail: samirb@synopsys.com).

E. Cerny is with the Département d'informatque et de recherche opérationnelle, Université de Montréal, Montréal, PQ, Canada H3C 3J7 (e-mail: cerny@iro.umontreal.ca).

B. Kaminska is with the Electrical Engineering Department, École Polytechnique de Montréal, PQ, Canada H3C 3A7 (e-mail: bozena@opmaxx.com).

B. Nadeau-Dostie is with LogicVision, Ottawa, Ont., Canada K1Z 8R3 (e-mail: benoit@lvision.com).

Publisher Item Identifier S 0278-0070(99)06631-2.

block would be obtained from a library. TM's are used to identify bits of signals/variables having too low controllability or observability. Test-point insertion is performed to improve controllability and observability, again at the RTL. Test points at the VHDL RTL are described by a set of synthesizable VHDL functions and procedures that insert control and observation points on bits of signals/variables. These functions/procedures are defined in a package which is included in the original specification.

We use a number of benchmark circuits which are random pattern resistant to show the effectiveness and the viability of the proposed method in terms of the resulting testability, area, and delay. The paper is organized as follows: Section II gives a summary of previous work in the literature. The overall approach is summarized in Section III. Section IV describes the DAG construction, while Section V presents the main formulas of TM's calculations. The test-point insertion method is discussed in Section VI. Section VII presents the experimental results, and Section VIII concludes the paper.

II. Previous Work

Recently, several RTL and behavioral level design and synthesis-for-testability approaches were proposed to generate easily testable circuits for partial scan, sequential ATPG, and BIST testing methodology [17]. The proposed approaches include RTL scan selection [7], [18], [19], modifications to the behavioral description of a design to improve the testability of the synthesized circuit [13], [20], and considering testability during the behavioral synthesis process [21]–[24]. The high-level techniques concentrate on improving the testability of datapaths, assuming that the controller can be tested independently and that its outgoing control signals to the datapath are fully controllable in the test mode [14], [25]. For hierarchical designs, a technique has been developed in [11], [15], and [20] to generate top-level test modes and constraints required to realize module's local test modes. The process of generating global test modes may reveal that some constraints cannot be satisfied, in which case, either the top level description or an individual module must be modified to satisfy the constraints.

Some RTL testability analysis methods were proposed to generate easily testable circuits for sequential ATPG [26], [27], [29]. The main objective of these methods is to reduce the ATPG CPU time at the expense of area overhead. In [26], an RTL testability method was proposed which allows testing using combinational test patterns. The methodology uses existing paths between registers, and with the help of multiplexers it loads combinational test patterns into the circuit flip-flops (FF's) without having to use scannable FF's. A technique was proposed later in [27] that can also use existing paths through functional units. Appropriate constants (identity elements) need to be added to the side inputs of the units to create I-paths [28]. Finally, an RTL testability analysis method was proposed in [5]. The authors use Verilog RTL models and functional verification patterns to improve the fault coverage of the resulting gate-level circuit. However, test-point insertion at the RTL was not addressed in this method and it was left as the designer's responsibility.

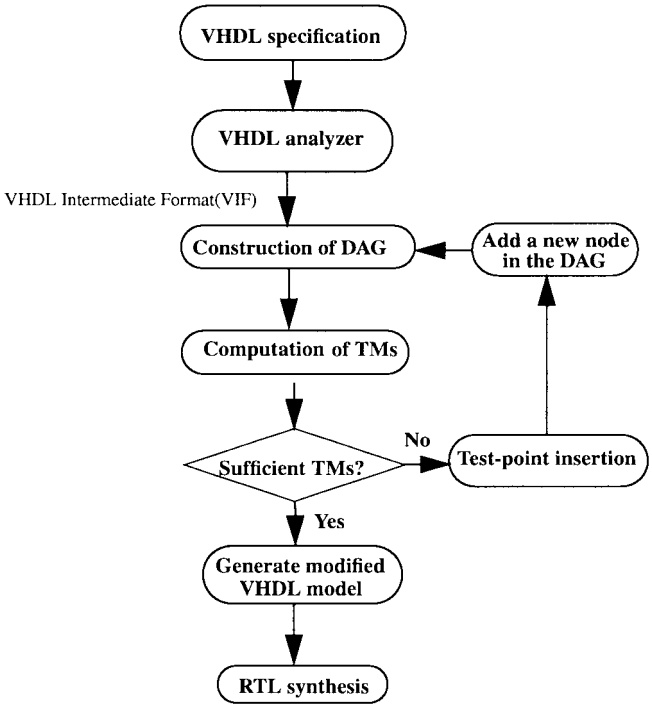


Fig. 1. Hardware synthesis with incorporated testability analysis and test-point insertion.

III. The Proposed Method

Fig. 1 depicts the overall structure of our testability analysis environment which can operate as a front-end to an RTL synthesis tool. In the first step, a VHDL analyzer from LEDA[1] is used to produce the VIF representation, and to identify all registers (full scan is assumed) and sequential VHDL statements. A DAG is used to store this information by linking the present states of registers with the next states through the VHDL statements. All integers are converted to bits or bit vectors and VHDL operations are modeled by their Boolean functional models. The TM's are the controllability and the observability of each bit of each signal/variable. They are then computed using this DAG by initializing the controllability of primary and pseudoprimary inputs to 0.5 (both zero and one), the observability of primary and pseudoprimary outputs to one, and by propagating them forward and backward through the VHDL statements.

The computed TM's allows us to identify hard-to-detect bits of signals/variables of the VHDL specification including the internal signals of FM's. This information is used to insert test points, again in the specification at the RTL by locally converting the affected signal/variable to the bit level and back. Each test point is described by a synthesizable VHDL function/procedure defined in a package. The function (procedure) is used to insert a control (observation) point on a given bit of a signal/variable in the VHDL specification and on internal signals of FM's. As a result of the test-point insertion, our method again produces a synthesizable RTL VHDL specification which can be input to an RTL synthesis tool. This allows designers to optimize their designs

[1]LEDA VHDL System, Version 4.0.3., LEDA S.A. 35 Avenue du Granier 38240 Meyland, France.

```
entity MOORE is              -- Moore machine
 port(X, CLOCK: in bit;
    Z: out BIT);
end;
architecture BEHAV of MOORE is
 type STATE_TYPE is (S0, S1, S2, S3);
 signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
 -- Process to hold combinational logic
 COMBIN: process(CURRENT_STATE, X)
 begin
  case CURRENT_STATE is
   when S0 =>
    Z <= '0';
    if X = '0' then
     NEXT_STATE <= S0;
    else
     NEXT_STATE <= S2;
    end if;
   when S1 =>
    Z <= '1';
    if X = '0' then
     NEXT_STATE <= S0;
    else
     NEXT_STATE <= S2;
    end if;
                          ../..
```

```
   when S2 =>
      Z <= '1';
      if X = '0' then
       NEXT_STATE <= S2;
      else
       NEXT_STATE <= S3;
      end if;
     when S3 =>
      Z <= '0';
      if X = '0' then
       NEXT_STATE <= S3;
      else
       NEXT_STATE <= S1;
      end if;
  end case;
end process COMBIN;
-- Process to hold synchronous elements (flip-flops)
 SYNCH: process
 begin
  wait until CLOCK = '1';
  CURRENT_STATE <= NEXT_STATE;
 end process SYNCH;
end BEHAV;
```

Fig. 2. VHDL specification of a Moore machine.

for different design constraints (e.g., area and delay) including testability. The algorithm in Fig. 1 was implemented in the C language in about 14 000 lines of code. In the following sections, we describe each step of the algorithm.

IV. Construction of the Direct Acyclic Graph

As shown in Fig. 1, a VHDL specification is compiled into its VIF representation, and then a DAG is constructed that represents the flow of information and data dependencies. Each internal node of the DAG corresponds to an operation of the VHDL specification such as arithmetic, relational, data transfer and logical operations. The source (sink) nodes of the DAG represent the present (next) state and primary inputs (outputs). The present and the next states are given by the registers that could be synthesized from the VHDL specification. Edges represent signals/variables declared by the designer in the specification and intermediate signals/variables as defined in Definition 1.

Note that no operation sharing is performed during the VHDL translation in the DAG. That means, each occurrence of a VHDL operation corresponds to a new node in the DAG.

Definition 1: An intermediate signal/variable is an unnamed signal/variable formed by an expression which is not a simple signal/variable name.

For example, in the expression $(A+B)+C$, there are two intermediate signals/variables: $(A+B)$ and $(A+B)+C$.

In the rest of this paper, whenever a signal/variable is mentioned, it refers to a signal/variable that is declared by the designer in the VHDL specification or to an intermediate signal/variable.

DAG Construction Steps:

1) generate control and data flow graph (CDFG) for each process of the VHDL specification;
2) unroll all for ... loops and expand procedures/functions by adding new nodes to the CDFG;
3) convert data types to bits;
4) translate the resulting CDFG's into a DAG.

Example 1: Consider the VHDL specification as shown in Fig. 2. It represents a Moore finite state machine with four states ($S0$, $S1$, $S2$, $S3$) and one output Z. The specification consists of two processes. Its DAG is shown in Fig. 3, the primary and pseudoprimary inputs/outputs representing the present and the next state registers are also indicated there. The signal *CURENT_STATE* is synthesized as a register and thus becomes a pseudoprimary two bit-wide input/output, since it is declared as an enumerated data type of four possible values. The constants $S0$, $S1$, $S2$, and $S3$ are encoded as 00, 01, 10, and 11, respectively. Each multiplexer corresponds to a conditional statement (*if-then-else*, *case*).

V. Testability Computation

Our testability analysis method handles most VHDL operations at the functional level in addition to logical operations. The following operators are supported by our method: n-bit adders, n-bit comparators, n-bit multipliers, n-bit subtractors,

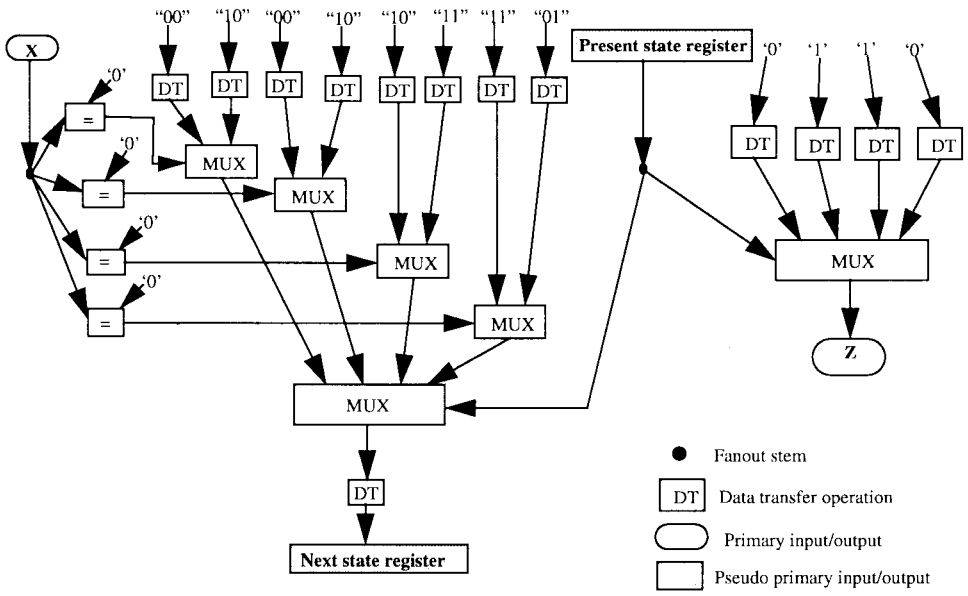


Fig. 3. DAG for TM's computation.

and multiplexers which are inferred by the conditional statements ("*if-then-else*," "*case*"). All of these are represented functionally, meaning that the controllability/observability propagation through them is computed with a high degree of accuracy. This is not the case with gate-level models, because reconvergent fanouts in such models may introduce errors in the calculations. Note that reconvergent fanouts at the RTL still remain between FM's and thus may introduce similar errors. However, the number of such reconvergent fanouts at the RTL is negligible when compared to the number of reconvergent fanouts at the gate level. In addition, it has been shown that most gate-level testability analysis tools still obtain good results within the presence of reconvergent fanouts. Thus, our testability analysis at the RTL can only be more accurate.

In our method we compute controllability of zero (C_0) and of one (C_1), and observability (O) values on each bit of each signal/variable and of internal signals of FM's. We show next the propagation of C_0, C_1, and O through typical VHDL operators.

Definition 2: Combinational controllability [2] is the probability that a signal s has a specific value. We have two measures, 1-controllability $(C_1(s))$ and 0-controllability $(C_0(s))$ such that $C_0(s) = 1 - C_1(s)$.

Definition 3: Combinational observability $O(l, s)$ of a line l on output s is defined as the probability that a signal change on l will result in a signal change on an output s. For multiple output modules, the observability of a line must be computed relative to each output and the overall observability $O(l)$ of l is computed based on formula given in (14) (Appendix A).

A. Controllability Calculations

In this section, we give the formulas for determining the controllability of an output of some VHDL operators, given the controllability of the inputs. We show next the controllability formula for an n-bit adder, the other formulas are included in Appendix A.

We wish to compute the controllability of the outputs of an n-bit adder, given the controllability of its inputs, assuming that the inputs are independent.

The 1-controllability measures $C_1(s_i)$ and $C_1(c_{i+1})$ can be computed by considering the minterms leading to a one on the respective output

$$C_1(s_i) = \alpha + C_1(c_i) - 2 \times \big(\alpha \times C_1(c_i)\big) \tag{1}$$

$$C_1(C_{i+1}) = \alpha \times C_1(c_i) + C_1(a_i) \times C_1(b_i) \tag{2}$$

where

$$\alpha = C_1(a_i) + C_1(b_i) - 2 \times C_1(a_i) \times C_1(b_i). \tag{3}$$

Note that α is the probability that $(a_i \oplus b_i) = 1$ and, consequently, $C_1(s_i)$ is the probability that $(a_i \oplus b_i \oplus c_i) = 1$.

To compute the controllability of each output of an n-bit adder, we can use a cascade of n full adders configured as a ripple-carry adder as shown in Fig. 4. There is no reconvergent fanouts in this circuit and all inputs are independent, hence the controllability computed on this tree structure is exact. To find the controllability measure $C_1(s_i)$ at the output s_i, (1)–(3) can be used for each 1-bit adder, and bit i can be evaluated when bits 0, 1, ..., $i-1$ have been computed. The calculation can be made in linear time in terms of the number of inputs. The same structure can be used to compute the controllability of a subtractor using 2's-complement representation of negation.

B. Observability Calculations

In this section, we give the observability formula for an n-bit adder. The other formulas can be found in Appendix A.

Consider again the n-bit ripple-carry adder shown in Fig. 4, and let us compute the observability of each input. According to the Boolean function of a 1-bit adder, the change on any input a_i, b_i, or c_i is always observable at s_i. It follows that:

$$O(a_i, s_i) = O(b_i, s_i) = O(c_i, s_i) = O(s_i), \quad i = 0, \ldots, n-1. \tag{4}$$

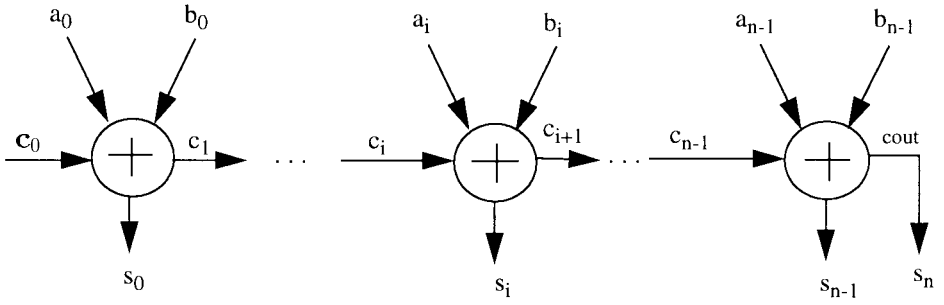


Fig. 4. A ripple-carry adder composed of n full adders.

The observability of an input at level i at the other outputs k, $k > i$, depends on the propagation of the carry from stage i to these outputs. For instance, to observe a_i at s_k such that $k = i + 1, \ldots, n$, we have to set $(b_i \oplus c_i) = 1$ and $(a_j \oplus b_j) = 1$, for all j such that $j = i+1, \ldots, k-1$. Equation (5) gives the general formula to compute the observability of each input a_i at outputs k, such that $k = i + 1, \ldots, n$

$$O(a_i, s_k) = \left[C_1(c_i \oplus b_i) \times \prod_{j=i+1}^{k-1} C_1(a_j \oplus b_j) \right] O(s_k), \quad i = 0, \ldots, n-1. \tag{5}$$

Similar formulas can be used to compute the $O(b_i, s_k)$ and $O(c_i, s_k)$ (in particular for c_0).

VI. Test-Point Insertion

In this section, we derive from the original VHDL specification a modified one that includes a set of test points expressed using synthesizable VHDL functions and procedures. Functions (procedures) are used to improve the controllability (observability) on bits of some signals/variables and the internal signals of FM's. The modified VHDL specification describes both the normal and the test modes of the given circuit.

Each test point to be inserted corresponds to a new node in the DAG. In turn, this node corresponds to a function/procedure to be added to the original VHDL specification. Therefore, we have to establish the relationship between the DAG representation and the VHDL specification. Each signal/variable candidate for test-point insertion is identified by a label, that is the name of the hierarchical path and the line number in the modified VHDL specification. The internal signal is identified by the line number in which the corresponding FM should have a test point inserted. At the VHDL level, control points consist of the logical *OR* or *AND* operations which combine as inputs a given bit of a given signal/variable and some extra control inputs that become part of the circuit inputs. The *OR* (*AND*) operation is used to increase the 1-controllability (0-controllability) value on the given bit of the given signal/variable. An observation point is implemented using a FF that loads the corresponding bit of the given signal/variable.

In the following, we first show how to insert test points on signals/variables in the VHDL specification and then on the internal signals of FM's. The algorithm used for selecting test points is presented after this section.

```
                                    ... /...
entity EXAMPLE is
  port(A, B        : in integer range 0 to 7;
       C, D        : in std_logic;
       E, F        : in std_logic_vector(3 downto 0);
       Z1          : out std_logic;
       Z2          : out std_logic_vector(3 downto 0);
       Z3          : out integer range 0 to 7
       );
end EXAMPLE;

architecture RTL of EXAMPLE is
begin
  process(A, B, C, D, E, F)
  begin
    -- Increase the 0-Controllability on signal Z1
    Z1 <= C + D;
    -- Increase the 1-Controllability on bit position 2 of signal Z2
    Z2 <= E and F;
    -- Increase the 1-Controllability on bit position 0 of signal Z3
    Z3 <= A + B;
  end process;
end RTL;
```

Fig. 5. VHDL example for controllability test-point insertion.

A. Test Points Insertion on Signals/Variables

In this section, we show how to insert control and observation points on bits of signals/variables in the VHDL specification. Functions (procedures) are used to insert control (observation) points. The functions/procedures are overloaded for different signal types and new parameters. The number of parameters depends on the data type of the corresponding signal/variable which can be either an integer, array of bits or a single bit.

1) Control-Point Insertion: Fig. 5 shows an example of a VHDL specification. Suppose that we want to insert three control points on signals $Z1$, $Z2$, and $Z3$ which are declared as three different data types. Assume that we want to increase the 0-controllability on single bit $Z1$, to increase the 1-controllability on bit position two of signal $Z2$ and to increase the 1-controllability on bit position zero of signal $Z3$. The modified VHDL specification including the required control points is shown in Fig. 6. To insert a control point, we use the same name of the function called *Insert_Control_Point()* which is defined in Fig. 6. For signal $Z1$, which is declared

```
. . .
Type CONTROL_POINT is (OR_POINT, AND_POINT);

entity EXAMPLE is
 port(A, B   : in integer range 0 to 7;
     C, D    : in std_logic;
     E, F    : in std_logic_vector(3 downto 0);
      -- The extra control points
     TEST_IN_1, TEST_IN_2, TEST_IN_3: in std_logic;
     -- Signal to switch between the test and the normal mode
     Test_Mode : in std_logic;
     Z1          : out std_logic;
     Z2          : out std_logic_vector(3 downto 0);
     Z3          : out  integer range 0 to 7);
end EXAMPLE;

-- Control point  insertion of a signal/variable
--  declared as STD_LOGIC
function Insert_Control_Point(
    SIG_VAR    :STD_LOGIC;
    CONT_TYPE  :CONTROL_POINT;
    TEST_IN    :STD_LOGIC;
    TM         :STD_LOGIC) return STD_LOGIC is
   variable V     : STD_LOGIC;
                        ../..
begin
  V := SIG_VAR;
  if (CONT_TYPE = OR_POINT) then
    V := (TM and TEST_IN) or V;
  else
    V := (not TM or TEST_IN) and V;
  end if;
  return (V);
 end Insert_Control_Point;
-- End of function definition

begin

process(A, B, C, D, E, F, TEST_IN_1, TEST_IN_2,
                                    TEST_IN_3, Test_Mode)
 begin
   Z1  <= Insert_Control_Point((C + D), AND_POINT,
                                    TEST_IN_1, Test_Mode);

   Z2  <= Insert_Control_Point((E and F), OR_POINT,
                                    2, TEST_IN_2, Test_mode);

   Z3  <= Insert_Control_Point((A + B), 3, OR_POINT,
                                    0, TEST_IN_3, Test_Mode);
 end process;
end RTL;
```

Fig. 6. Modified VHDL specification including three control points.

as a single bit (std_logic), we need as function parameters the expression assigned to the signal, the type of the control point to insert (*AND* operation), an extra control input (*TEST_IN_1*),[2] and the signal *Test_Mode* which is used to switch from the normal mode to the test mode and vice-versa. The *AND* operation combines the output bit of the expression, the control input *TEST_IN_1* and the signal *Test_Mode*. The function is defined here in the architecture, but usually it would be placed in a package. The constant signal *CONT_TYPE* declared as an enumerated type is used to select between the control point type. In Fig. 6, we showed only the function definition to insert a control point on signal $Z1$ of type std_logic.

The same overloaded function name is used to insert a control point on signals $Z2$ and $Z3$ which are of different types. For signal $Z2$, another parameter is used to specify the bit position and for signal $Z3$, we need a parameter which is used to specify the number of bits required to convert the integer type.

2) Observation-Point Insertion: Fig. 7 shows another VHDL example. It consists of one unclocked process with two sequential assignment statements. Assume now that we want to increase the observability value on bit position two of variable V. An observation point is thus required on this bit position. This consists of attaching a FF to this specific bit position and loading it by that bit value. The modified VHDL specification is shown in Fig. 8. A procedure called *Insert_Observation_Point()* is used to achieve this effect at the VHDL level. Bit 2 of V is first transferred to an internal signal S. Next, signal S is transferred to a signal *SCAN_OUT* which is assigned in a clocked process in order to infer a register.

[2]This signal can come from a register which will be included in the scan chain.

```
                                    ... / ...
entity EXAMPLE is
 port (A, B, C          : in std_logic_vector(3 downto 0);
     Z                  : out std_logic_vector(3 downto 0));
end EXAMPLE;

architecture RTL of EXAMPLE  is
begin
 process(A,B,C)
 variable V      : std_logic_vector(3 downto 0);
 begin
  -- Insert an Observation point on bit position 2 of assigned variable V
   V := A and B;
   Z <= V or C;
 end process;
end RTL;
```

Fig. 7. VHDL example for observability test-point insertion.

Note that the same procedure name is again overloaded to observe any bit of any data type signal/variable.

B. Test-Point Insertion on Internal Signals of FM's

In the previous section, we only analyzed the testability of signals/variables that are explicitly declared or may be implied in the VHDL specification. These signals/variables are used to connect VHDL operators which are mapped to FM's (adders, comparators, etc.). It is well known that large multibit FM's can be difficult to test by random patterns due to low controllability/observability of their internal signals such as the carry lines inside an adder. These signals do not have a direct correspondence in the VHDL specification for test-point insertion as in the previous examples. In the following, we show how we insert test points at the RTL on such internal signals.

```
. . .
entity EXAMPLE is
 port (A, B, C          : in std_logic_vector(3 downto 0);
       Z                : out std_logic_vector(3 downto 0);
       CLK              : in bit;
    -- Signal used for observation
    SCAN_OUT     : out std_logic);
end EXAMPLE;

architecture RTL of EXAMPLE is
-- Observation point insertion of a signal/variable
--  declared as STD_LOGIC_VECTOR
   Procedure Insert_Observation_Point(
    SIG_VAR     :STD_LOGIC_VECTOR;
    POSITION    :INTEGER;
    signal SCAN  :out STD_LOGIC ) is
 begin
  SCAN <= SIG_VAR(POSITION);
 end Insert_Observation_Point;

 signal S     : std_logic;
                . . /. .
```

```
begin
 process(A, B, C)
 variable V : std_logic_vector(3 downto 0);
 begin
   V := A and B;
  -- Insert an observation point at bit 2 of variable V
  Insert_Observation_Point(V, 2, S);
  Z <= V or C;
 end process;

 -- This clocked process is used to infer registers for observation points
 process
 begin
  wait until CLK = '1';
  SCAN_OUT <= S;
 end process;
end RTL;
```

Fig. 8. Modified VHDL specification including one observation point.

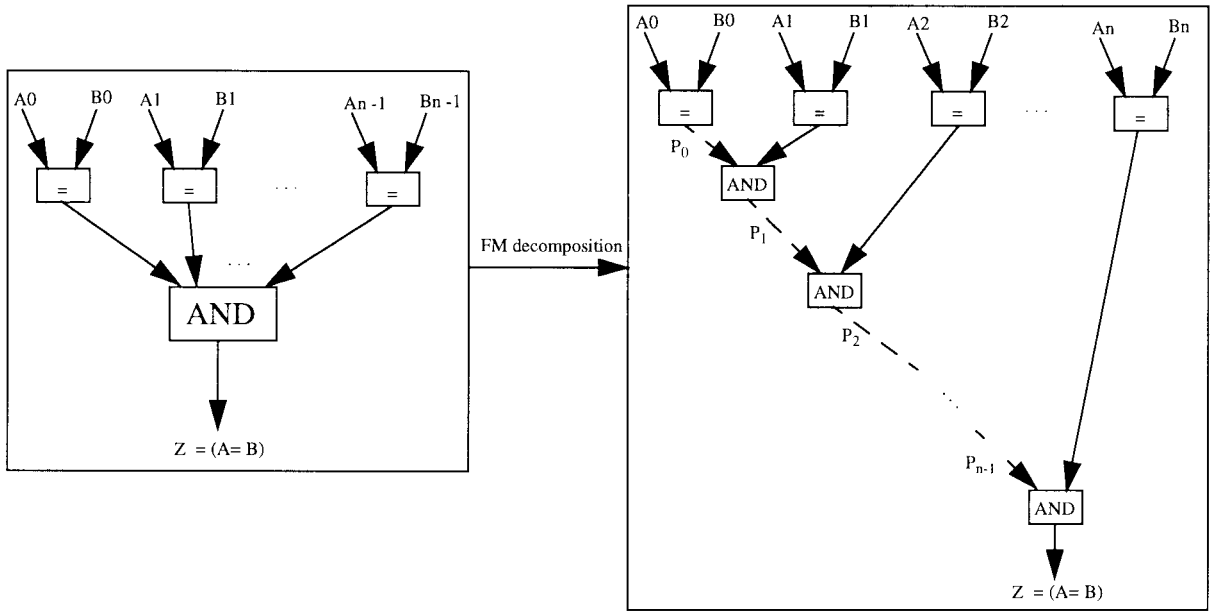


Fig. 9. (a) The structure of an n-bit equality comparator and (b) the internal signals considered for test-point insertion.

1) Comparators: As an example, consider the equality comparator. An n-bit equality comparator can be decomposed functionally into n 1-bit equality comparators. It is known that the n-input *AND* operation is difficult to test with random testing when the number of inputs is large. One solution is to decompose the n-input *AND* operation into a cascade of $(n-1)$ two-input *AND* operations as shown in Fig. 9(b). The signals shown as dotted lines are considered as internal signals and they are not visible in the original VHDL specification. To make these signals visible, we decompose the comparator into smaller comparator blocks. The number of comparator blocks depends on the number of internal signals having low controllability/observability. Note that each block of the comparator would still be obtained from the library and need not to be resynthesized. Here also, we defined a function (procedure) which inserts a given number of control (observation) points at some internal signals of an equality comparator. Its use will be illustrated next on an example.

Fig. 10 shows a VHDL specification of a 16-bit counter in which a 16-bit equality comparator is used in the "if" expression. We want to insert two control points at two different internal signals of the comparator. For example, we want to increase the 1-controllability on both signals. Each internal signal is characterized by a position in the equality comparator structure. The signal P_i as shown in Fig. 9(b) corresponds to comparator i and is assigned the ith

```
                    . . .
entity COMP is
 port(CLEAR          : in std_logic;
    IN_COUNT         : in std_logic_vector(15 downto 0);
    CLOCK            : in bit;
    OUT_COUNT        : out std_logic_vector(15 downto 0)
    );
end COMP;

architecture RTL of COMP is
begin
 process
 begin
  wait until CLOCK = '1';
  if (CLEAR = '1' or (IN_COUNT = "1111111111111111")) then
       OUT_COUNT <= "0000000000000000";
  else
      OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Fig. 10. A VHDL specification of a 16-bit counter.

position in the comparator structure (counting from zero). This information is a parameter in the function (procedure) used to insert a control (observation) point at a specific internal signal, P_i. The modified VHDL specification including the required control points at positions five and ten, is shown in Fig. 11. The function *Insert_Control_Equal()* is used to insert a given number of control points at internal signals of the equality comparator. The function takes the following parameters: The two operands of the comparison L and R, the number of control points to insert N, the corresponding positions of the control points represented by the array P, the type of the control points (increasing the 1-controllability or the 0-controllability), the required extra control inputs as an array *Sig_array* and the signal *Test_Mode*. Note that this functions calls the function *Insert_Control_Point()* defined previously. Both of these functions are defined in the package "*Test_Points*" which is included in the specification as shown in Fig. 11. In the same manner, we define a procedure called *Insert_Observation_Equal()* which is used to insert a given number of observation points on internal signals of the equality comparator.

We can use a similar method as for the equality comparator to insert test points on internal signals of the other comparator types ($<$, $>$, $<=$, $>=$). For instance, the logic equation of an n-bit less than comparator is given in (9) in Appendix A. This comparator requires an n-input *OR* operation which can be decomposed into a cascade of $(n-1)$ two-input *OR* operations.

2) Adders: In an adder, the carry out lines (Fig. 4) are the internal signals to consider for test-point insertion. Test-point insertion inside multiplier FM's is implemented based only on the shift-add structure. Other implementations of the multiplier are under investigation.

3) Multiplexers: Multiplexers are inferred by conditional statements ("*if-then-else*," "*case*") in the VHDL specification. Fig. 12(a) shows a VHDL specification containing a "*case*" statement. The corresponding DAG representation is given in Fig. 12(b). A $n:1$ multiplexer (mux) can be difficult to test when the number of inputs (i.e., the number of cases in the "*case*" statement) increases, as is often the case in VHDL designs. This difficulty comes from the internal signals that are not visible at the VHDL level. Fig. 13 shows a possible representation of a $4:1$ mux by means of $2:1$ muxes. The internal signals are shown as dotted lines. Test points inserted on the internal signals do not have a direct correspondence in the VHDL specification. One solution is to decompose the large case statement into smaller nested case statements. However, this may be difficult and can dramatically change the original VHDL code. Another solution consists of using the inputs and the outputs of the $4:1$ mux to improve controllability and observability values of the internal signals. In fact, we can increase the observability of the mux output which is visible in the VHDL specification. We thus insert an observation point on variable V which is the output of the $4:1$ mux. We use the same approach as defined in Section VI-A to insert an observation point. This corresponds to adding the procedure *Insert_Observation_Point()* just after the line corresponding to the end "*case*" statement (see Fig. 14).

C. Test Point Selection Algorithm

In this section, we present a greedy algorithm used for selecting test points (control and observation) [6], [14]. However, any more efficient test-point insertion method can be used to select the best locations for insertion. Before we can describe the algorithm, we give some definitions.

Definition 4 [4]: The detectability of a fault in a single-bit signal S is defined as follows.

When S is stuck-at 1, we have

$$D_1(S) = (1 - C_1(S)) \times O(S). \tag{6}$$

Similarly, if S is stuck-at-0, then we have

$$D_0(S) = C_1(S) \times O(S). \tag{7}$$

Definition 5: We define a VHDL specification as random testable if each bit of each signal/variable and of each internal signal of FM's has a Detectability value above a given threshold Dth. The value of Dth comes from experience and depends on the desired fault coverage, test length, and circuit complexity. Detectability below the Dth value indicates potential controllability and/or observability problems that may negatively impact the length of the test sequence of the resulting circuit.

The test-point insertion process starts with the calculation of the Detectability value of each bit of each signal/variable in the VHDL specification and the internal signals of FM's. Among the signals and variables with Detectability values below a Dth, we first select those that have controllability values below the controllability threshold. The candidates that are nearest to the primary or the pseudoprimary inputs are selected; this is because control point insertion there will affect the controllability values not only of the insertion point, but also of all the signals/variables driven by it. Once the insertion is done, the controllability values are recomputed, and the process is repeated until all controllability values are above the threshold.

```
. . .
use WORK.Test_Points.all;

entity COMP is
 port(CLEAR                    : in std_logic;
    IN_COUNT                   : in std_logic_vector(15 downto 0);
    CLOCK                      : in bit;
    TEST_IN_1, TEST_IN_2       : in boolean;
    Test_Mode                  : in std_logic;
    OUT_COUNT                  : out std_logic_vector(15 downto 0)
    );
end COMP;

architecture RTL of COMP is
type Position_array is array(INTEGER range <>) of INTEGER;
type Cont_type_array is array(INTEGER range <>) of CONTROL_POINT;
type Sig_array is array(INTEGER range <>) of BOOLEAN;

function unsigned_Insert_Control_Equal(L,R :UNSIGNED; N:INTEGER; P:POSITION_ARRAY;
CONT_TYPE:Cont_TYPE_ARRAY; TEST_IN:SIG_ARRAY;TM : STD_LOGIC ) return BOOLEAN is
 variable V:BOOLEAN;
begin
 V := (L(L'left downto (P(0)+1+L'right)) = (R'left downto (P(0)+1+R'right)));
For i in (N-1) downto 0 loop
  if (i = N-1) then
    V := V and (Insert_Control_Point((L((P(i) + L'right) downto L'right) =
                                R((P(i) + R'right) downto R'right)), CONT_TYPE(i), TEST_IN(i), TM));
  else
    V := V and (Insert_Control_Point ((L((P(i) + L'right) downto (P(i+1) + 1 +L'right)) =
                                R((P(i) + R'right) downto (P(i+1) + R'right + 1))),CONT_TYPE(i), TEST_IN(i), TM));
 end if;
 end loop;
 return V;
end unsigned_Insert_Control_Equal;

function Insert_Control_Equal(L,R: STD_LOGIC_VECTOR; N: INTEGER; P: POSITION_ARRAY;
CONT_TYPE: Cont_TYPE_ARRAY; TEST_IN: SIG_ARRAY;TM: STD_LOGIC) return BOOLEAN is
begin
 return unsigned_Insert_Control_Equal(UNSIGNED(L), UNSIGNED(R), N, P, CONT_TYPE, TEST_IN, TM);
end Insert_Control_Equal;

begin
 process
 begin
  wait until CLOCK = '1';
 if (CLEAR = '1' or (Insert_Control_Equal (IN_COUNT, "1111111111111111", 2, (10, 5),(OR_POINT, OR_POINT),
                          (TEST_IN_1,TEST_IN_2), Test_Mode))) then
   OUT_COUNT <= "0000000000000000";
 else
   OUT_COUNT <= IN_COUNT + "0000000000000001";
  end if;
 end Process;
end RTL;
```

Fig. 11. Modified VHDL specification of a 16-bit counter: control point insertion on the internal signals of an equality comparator.

Next, the analysis considers observability values. Decisions to improve observability are deferred because a change in controllability may affect observability, but a change in observability has no effect on controllability. If control points are inserted, the observability values are recomputed on all bits of all signals/variables and internal signals. Those with observability values below the observability threshold are candidates for observation point insertion; among them, the ones closest to the primary and the pseudoprimary outputs are selected first. Once observation point insertion is done, the observability values are recomputed, and the process is repeated until all observability values are above the threshold.

VII. Experimental Results

In this section, we present the experimental results obtained on three benchmark circuits which are random pattern resistant. Circuits $C1$ and $C2$ (Table I) are some design blocks in an input–output chip[3] that were designed at the RTL and synthesized by *Synopsys* tools to the gate-level. Circuit $C3$

[3] Private Benchmark Circuits, LogicVision, Ottawa, Ont., Canada.

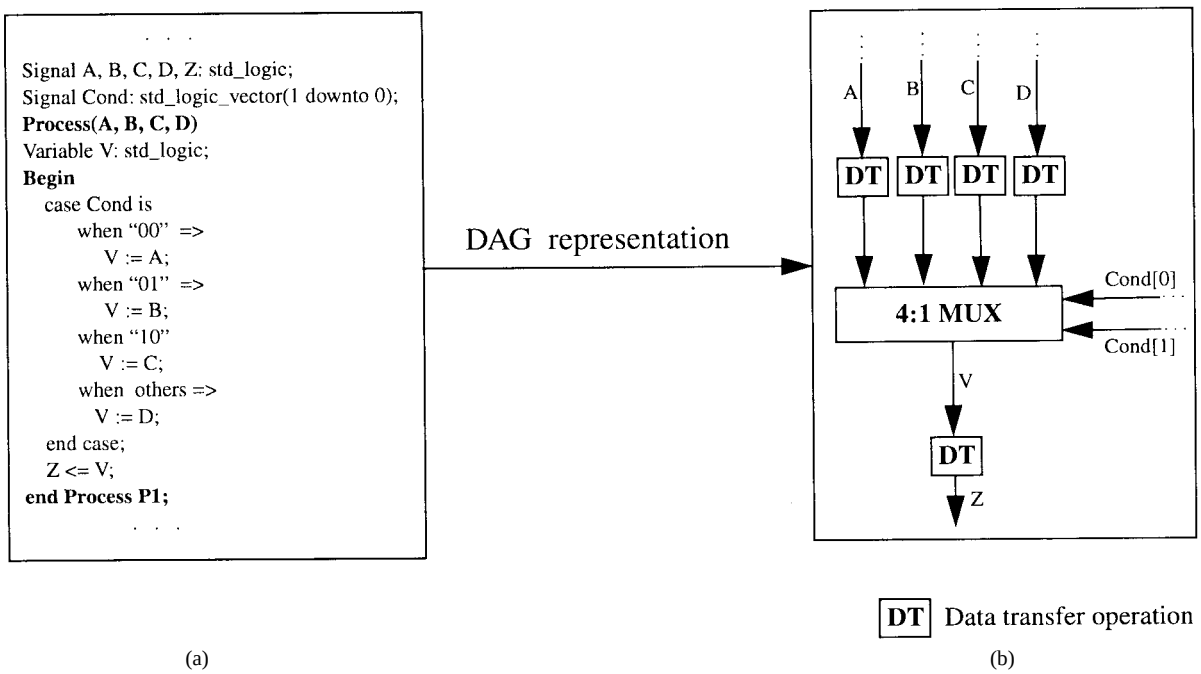


Fig. 12. (a) A VHDL "*case*" statement. (b) DAG representation.

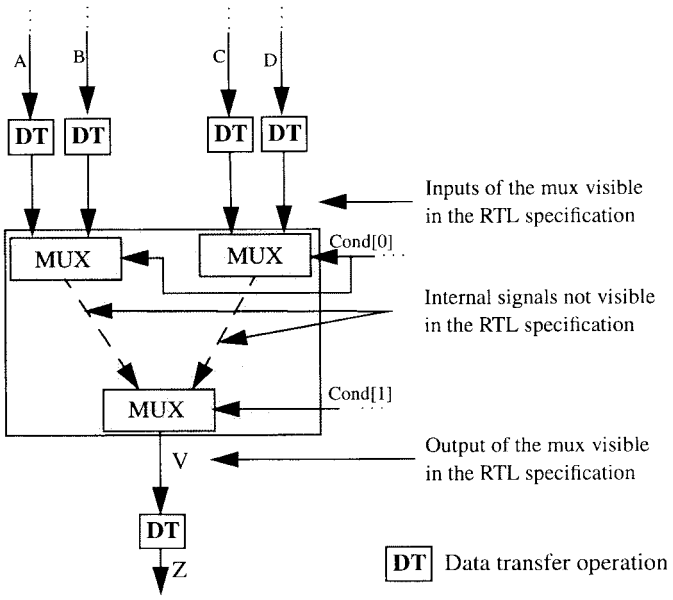


Fig. 13. Internal signals of a 4:1 mux.

is 32-bit counter. All circuits contain both control and data operations. The largest circuit is $C2$ which consists of about 1200 VHDL lines.

Each circuit is synthesized to the gate-level before and after the test-point insertion. A random ATPG tool is used to evaluate the fault coverage of the gate level circuit with the full scan option before and after the test-point insertion. For each circuit we fix the Detectability threshold value Dth depending on the circuit complexity and the test length, and then determine the required test points. For the three circuits under consideration, the Detectability threshold is set to 0.001, the controllability threshold is set to 0.01, and the observability threshold to 0.001. The circuits were synthesized under various area and delay constraints. The delay is the worst case propagation path in the resulting gate-level circuit.

```
. . .
Procedure Insert_Observation_Point(
   SIG_VAR     :std_logic;
   signal SCAN  :out std_logic ) is
 begin
  SCAN <= SIG_VAR;
 end Insert_Observation_Point;

Signal A, B, C, D, Z, SCAN: std_logic;
Signal Cond: std_logic_vector(1 downto 0);

Process(A, B, C, D)
Variable V: std_logic;
Begin
  case Cond is
      when "00" =>
         V := A;
      when "01" =>
         V := B;
      when "10"
         V := C;
      when others =>
        V := D;
  end case;
   Insert_Observation_Point(V, SCAN);
  Z <= V;
end Process P1;

-- This clocked process is used to infer registers for observation points
 process
 begin
  wait until CLK = '1';
  SCAN_OUT <= SCAN;
 end process;
```

Fig. 14. Modified VHDL specification with one observation point to enhance the detectability of the internal signals of a mux.

TABLE I
CIRCUIT INFORMATION

Circuit	No. of inputs	No. of outputs	No. of Flip-Flops
C1	3	17	16
C2	21	50	31
C3	33	32	32

Tables II and III show the results obtained under two different area and delay constraints. They are identified as Optimization 1 (Table II) and Optimization 2 (Table III). In each case, we performed a fault simulation of 32K random patterns before and after the test-point insertion. We show the following quantities in the tables: the resulting fault coverage after applying 32K random patterns, the number of test points (control/observation points) added to the VHDL specification, the total cell area, the delay of the critical path incurred by the test-point insertion, the corresponding percentage area, the worst delay and the total number of faults over the number of redundant faults. Note that each unit of the total cell area corresponds approximately to a two-input gate.

Considering the results of Tables II and III, we can see that, first, the insertion of a small number of test points leads to an increased fault coverage after applying 32K random patterns. This indicates that our method provides good analysis of testability, even though it is carried out at the RTL, while the coverage analysis is done at the gate level. For example, in the case of $C1$, inserting five test points improves the fault coverage from 95.48 to 100% (86.98 to 99.76%) in the case of Optimization 1 (Optimization 2). Similarly, remarkable improvements can be observed for the other two circuits. The area overheads range from 3%–10%. However, the delays actually improved after test-point insertion for $C1$ and $C2$ as shown in Table II, and for $C2$ as shown in Table III. This indicates that the insertion does not necessarily imply an increase in area and delay if carried out at the VHDL source code. The synthesis tool optimizes concurrently the inserted test points and the functional logic within the design constraints (delay and area). This is one of the main advantages of inserting testability at the RTL before synthesis. Designers have complete control of the overall optimization process and the test points remain an integral part of the RTL specification.

Note that we can still obtain an improvement by modifying the optimization constraints. Circuit $C2$ is the highest random pattern resistant circuit for which we obtained very good results in terms of delay at the expense of a small increase in area (approximately 3%). The fault coverage increased approximately by 15% for this circuit. Note that the circuit required ten test points inside the internal signals of some of the equality comparators implied in the VHDL specification.

Another important criterion characterizing random pattern testability of a circuit is the test length required to achieve a certain fault coverage. For each of the circuits considered, Table IV lists the test length which is necessary to obtain a certain fault coverage before and after test-point insertion. We consider only the constraints used in Optimization 1. It can be seen that the test length can be reduced by several orders of magnitude with very few test points. Since, the test length is proportional to the time required to perform the self-test, a similar reduction in the test time is achieved. We can notice that for $C2$ which is highly random test resistant, 95% fault coverage is achieved with 19 test points. The maximum fault coverage before test-point insertion is less than 80% for 10^7 random patterns, while we achieve 95% after test-point insertion with only 2×10^5 random patterns.

VIII. CONCLUSIONS

A new testability analysis and test-point insertion method for RTL VHDL designs was proposed. We analyze and modify the VHDL RTL description of the circuit, to generate an easily testable gate-level circuit by a pseudorandom vectors in a BIST environment. We identify hard-to-detect bits of signals/variables that are explicitly declared or implied in the VHDL specification. Test-point insertion is carried out again at the RTL. Internal signals of FM's are also analyzed and may be modified at the RTL. Test points are defined using a set of overloaded VHDL functions and procedures defined in a package. Since the inserted test points are included in the original VHDL code, they become part of the specification before synthesis. This allows full use of RTL synthesis tools to optimize both the functional and the inserted test logic together within the required design constraints (delay and area) for a given technology. In fact, the performances of the gate-level circuit can be the same with or without the insertion of test points. Another advantage of our method when compared to other existing methods [7], [13], [14] is that we can affect each bit of each signal/variable regardless its type (integer, bit vector, single bit). A number of random-pattern-resistant benchmark circuits were used to demonstrate the effectiveness of our method.

APPENDIX
CONTROLLABILITY AND OBSERVABILITY COMPUTATION

In this Appendix, we give the main formulas for computing the controllability and observability of some VHDL operators at the functional level. The controllability formulas for logical operators can be found in [3]. Note that the controllability (observability) of the adder was given in Section V-A (Section V-B).

A. Controllability Calculation

1) Controllability of the Outputs of an n-Bit Multiplier: Given the controllability measures of the inputs, the exact controllability measures on a n-multiplier outputs can be found by performing 2^{2n} operations. The computation of the probability of occurrence of each number A ($0 \leq A < 2^n - 1$) is based on the controllability measure on each input bit. The method based on the truth table is exact for an n-bit multiplier with $n \leq 10$. However, when n exceeds ten, the memory requirements and the CPU time can become very excessive. One solution to reduce this complexity is to decompose the multiplier into a number of smaller multipliers. For example, an n-bit multiplier can be formed using four m-bit multipliers

TABLE II
EXPERIMENTAL RESULTS WITH OPTIMIZATION 1

Circuit	Fault coverage [%] after 32 K random patterns		Number of Test Points	Total cell area (units)			Delay (ns)			No. of faults/No. of Redundant faults	
	Before insertion	After insertion	Contr. P/ Obs. P	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion
C1	95.48	100	4/1	250	266	6.40	12.85	12.65	-1.58	790/0	888/0
C2	76.20	90.40	14/5	967	996	3.00	46.75	43.43	-7.10	3760/0	4155/0
C3	87.81	92.75	5/0	451	460	1.99	17.81	17.81	0.00	1186/52	1362/0

TABLE III
EXPERIMENTAL RESULTS WITH OPTIMIZATION 2

Circuit	Fault coverage [%] after 32 K random patterns		Number of Test Points	Total cell area (units)			Delay (ns)			No. of faults/No. of Redundant faults	
	Before insertion	After insertion	Contr. P/ Obs. P	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion	Ovhd. [%]	Before insertion	After insertion
C1	86.98	99.76	4/1	399	439	10.02	10.59	11.67	10.19	1588/0	1799/0
C2	75.56	89.79	14/5	930	964	3.65	46.29	42.13	-9.98	3632/0	3780/0
C3	84.60	93.48	5/0	473	509	7.61	17.48	17.48	0.00	1404/32	1580/0

TABLE IV
EXPERIMENTAL RESULTS: COMPARISON OF TEST LENGTH WITH OPTIMIZATION 1

Circuit	Before insertion		After insertion	
	Test length	Max fault coverage[%]	Test length	Max fault coverage[%]
C1	800,000	100	8,000	100
C2	> 10,000,000	80.00	200,000	95.00
C3	> 10, 000, 000	91.31	200,000	94.62

(with $m = n/2$) as compactly described in Fig. 15. This method does not compute the exact controllability but gives, in our opinion, a good measure of controllability.

2) Controllability of the Output of an n-Bit Comparator: We compute the controllability of $(A = B)$ and $(A < B)$, and deduce the controllability of the other ones. Let the two n-bit numbers to be compared have the form: $A = A_{n-1}A_{n-2} \cdots A_0$ and $B = B_{n-1}B_{n-2} \cdots B_0$.

The controllability of the output of an n-bit equality comparator is as follows:

$$C_1(A = B) = \prod_{i=0}^{n-1} C_1(A_i = B_i). \tag{8}$$

The logic equation for $(A < B)$ may be written as

$$\begin{aligned}(A < B) = \big((A_{n-1} < B_{n-1}) \vee (A_{n-1} = B_{n-1}) \\ \wedge (A_{n-2} < B_{n-2})\big) \cdots \\ \vee \big((A_{n-1} = B_{n-1}) \wedge (A_{n-2} = B_{n-2}) \\ \wedge \cdots \wedge (A_1 = B_1)\big) \wedge (A_0 < B_0).\end{aligned} \tag{9}$$

Thus, 1-controllability of $(A < B)$ is given by the following formula:

$$\begin{aligned}&C_1(A < B) \\ &\quad = C_1(A_{n-1} < B_{n-1}) + \sum_{i=1}^{n-1} \\ &\qquad \times \left(C_1(A_{n-1-i} < B_{n-1-i}) \times \prod_{j=1}^{i} C_1(A_{n-j} = B_{n-j})\right).\end{aligned} \tag{10}$$

3) Controllability of the Output of a Multiplexer: Consider a multiplexer of n control inputs $(c_0c_1 \cdots c_{n-1})$ and 2^n possible data inputs, $(A_0A_1 \cdots A_{2^n-1})$ in which each input A_i and the corresponding output consist of m bits. The general formula to compute the controllability at the output is as follows:

$$C_1(S(k)) = \sum_{i=0}^{2^n-1} C_1\big(A_i(k)\big) \times C_1\big(\text{control}, A_i(k)\big) \tag{11}$$

where $k = 0, 1, \ldots, m-1$, $i = 0, 1, \ldots, 2^n - 1$. $C_1(\text{control}, A_i(k))$ is the 1-controllability on the control inputs such that data input $A_i(k)$ is selected to the output $S_i(k)$.

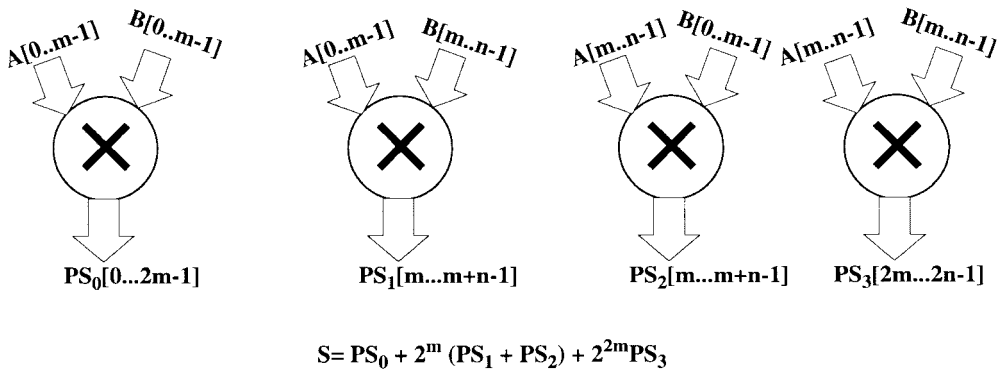


Fig. 15. n-bit multiplier composed of four m-bit multiplier ($m = n/2$).

4) Controllability of a Fanout Stem: A fanout stem is a point where a driving signal is connected to more than one combinational or register inputs, these inputs are then the fanout branches.

The controllability of a fanout branch is equal to the controllability of its stem.

B. Observability Calculation

1) Observability of the Inputs of an n-Bit Multiplier: To compute the observability of the multiplier, we use the same method used for computing the controllability of the inputs. For an n-bit multiplier with $n <= 10$, we use the corresponding truth table to compute the observability of each bit input. However, when n is over ten, we decompose the multiplier into small blocks as shown in Fig. 15. Then, the observability is propagated through multiplier and adder blocks.

2) Observability of the Inputs of an n-Bit Comparator: Consider a comparator with two inputs A and B of the form $A = A_{n-1}A_{n-2}\cdots A_0$ and $B = B_{n-1}B_{n-2}\cdots B_0$, and output S.

The observability of each input A_i or B_i at the output S of the comparators ($A = B$) and ($A \neq B$) is computed using (12). The same equation can be used to compute the observability of B_i by interchanging B_i for A_i

$$O(A_i, S) = \left[\prod_{j=0/i\neq j}^{n-1} C_1(A_j = B_j)\right] \times O(S). \qquad (12)$$

To compute the observability of inputs of the ($A < B$) comparator, let $A_+ = A_{n-1}A_{n-2}\cdots A_{i+1}$ and $A_- = A_0A_1\cdots A_{i-1}$. Define B_+ and B_- in a similar fashion. The operation of this comparator is not commutative and thus different formulas are used to compute the observability of A and B. To observe the input bit A_i, we have to maintain $(A_+ = B_+)$ and depending on the value of bit B_i we have to inspect the relation between A_- and B_-.

The observability of A_i is computed using the following equations:

$$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times \alpha$$
$$\text{for } i = 1, 2, \cdots, n-2, \text{ where}$$
$$\alpha = C_1(A_- \geq B_-) \times C_1(B_i) + C_1(A_- < B_-) \times \big(1 - C_1(B_i)\big)$$

$$O(A_i, S) = O(S) \times C_1(A_+ = B_+) \times C_1(B_i) \quad \text{for } i = 0$$

and

$$O(A_i, S) = O(S) \times \alpha, \qquad \text{for } i = n-1, \text{ where}$$
$$\alpha = C_1(A_- \geq B_-) \times C_1(B_i) + C_1(A_- < B_-) \times \big(1 - C_1(B_i)\big).$$

By the same analysis, we can compute the observability values of the other comparator types.

3) Observability of Multiplexer Inputs: Consider again a multiplexer consisting of n control inputs ($c_0c_1\cdots c_{n-1}$) and 2^n data possible inputs, ($A_0A_1\cdots A_{2^n-1}$), in which each input A_i and the output consist of m bits. The observability computation for the data input is

$$O\big(A_i(k)\big) = O\big(S(k)\big) \times C_1(\text{control}, A_i(k))$$

where $C_1(\text{control}, A_i(k))$ is the 1-controllability of the control inputs such that $A_i(k)$ is connected to $S_i(k)$,

$$k = 0, 1, \cdots, m-1 \quad \text{and} \quad i = 0, 1, \cdots, 2^n - 1.$$

The observability of a control input c_i is computed using the following formula:

$$O(C_i) = \text{Max}_k\left(\sum_{p=0}^{2^{n-i-1}-1}\left(\sum_{j=p\times 2^{i+1}}^{p\times 2^{i+1}+2^i-1} C_1(\beta_j)\right) \times C_1(\text{control}, \beta_j)\right) \times O\big(S(k)\big)\Bigg) \qquad (13)$$

where $k = 0, 1, \ldots, m-1$, $\beta_j = [A_j(k) \oplus A_{j+2^i}(k)]$, and $C_1(\text{control}, \beta_j)$ is the 1-controllability of the control inputs such that the inputs $A_j(k)$ and $A_{j+2^i}(k)$ are mutually exclusive, i.e., $\beta_j = 1$.

4) Observability Computation of a Fanout Stem [3]: The observability of a fanout stem, s, is related to the observability at each fanout branch of the fanout stem. Let n be the number of fanout branches for a fanout stem and let $O(b_i)$ be the observability value at the ith branch. Then the observability of the fanout stem is computed using the following equation:

$$O(s) = 1 - \prod_{i=1}^{n}\big(1 - O(b_i)\big). \qquad (14)$$

REFERENCES

[1] L. H. Goldstein, "Controllability/observability analysis," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 685–693, Sept. 1979.

[2] C. H. Chen and P. R. Menon, "An approach to functional level testability analysis," in *Proc. Int. Test Conf.*, Oct. 1989, pp. 373–379.

[3] P. H. Bardel, W. H. McAnney, and J. Savir, *Built-In Test for VLSI Pseudorandom Techniques.* New York: Wiley, 1987.

[4] B. H. Seiss, P. M. Trouborst, and M. H. Schlz, "Test-point insertion for scan-based BIST," in *Proc. European Test Conf.*, Apr. 1991, pp. 253–262.

[5] W. Mao and R. K. Gulati, "Improving gate-level fault coverage by RTL fault grading," in *Proc. Int. Test Conf.*, Oct. 1996, pp. 463–472.

[6] M. Youssef, Y. Savaria, and B. Kaminska, "Methodology for efficiently inserting and condensing test points," *Proc.-E Inst. Elect. Eng.*, vol. 140, no. 3, pp. 154–160, May 1993.

[7] X. Gu *et al.*, "Testability analysis and improvement from VHDL behavior specification," in *Proc. European Test Conf.*, Mar. 1994, pp. 644–649.

[8] C. H. Chen and D. G. Saab, "Behavioral synthesis for testability," in *Proc. Int. Conf. Compter-Aided Design*, Nov. 1992, pp. 612–615.

[9] S. Bhattacharya, F. Brglez, and S. Dey, "Transformation and resynthesis for testability of RTL control-data path specifications," *IEEE Trans. VLSI Syst.*, vol. 1, no. 3, pp. 304–318, Sept. 1993.

[10] S. Dey and M. Potkonjak, "Transforming behavioral specifications to facilitate synthesis of testable designs," in *Proc. Int. Test Conf.*, Oct. 1994, pp. 184–193.

[11] P. Vishakantaiah, T. Thomas, J. A. Abraham, and M. Abadir, "AMBIANT: Automatic generation of behavioral modifications for testability," in *Proc. Int. Conf. Computer Design*, Oct. 1993, pp. 63–66.

[12] C. H. Chen and D. G. Saab, "A novel behavioral testability measure," *IEEE Trans. Computer-Aided-Design*, vol. 12, pp. 1960–1970, Dec. 1993.

[13] H. Chen, T. Karnik, and D. G. Saab, "Structural and behavioral synthesis for testability techniques," *IEEE Trans. Computer-Aided-Design*, vol. 13, pp. 777–785, June 1994.

[14] C. A. Papachristou and J. Carletta, "Test synthesis in the behavioral domain," in *Proc. Int. Test Conf.*, Oct. 1995, pp. 693–702.

[15] P. Vishakantaiah, J. A. Abraham, and M. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," in *Proc. 29th Design Automation Conf.*, June 1992, pp. 273–278.

[16] C. H. Cho and J. Armstrong, "B-algorithm: A behavior test generation algorithm," in *Proc. Int. Test Conf.*, Oct. 1994, pp. 968–979.

[17] K. D. Wagner and S. Dey, "High-level synthesis for testability: A survey and perspective," in *Proc. Design Automation Conf.*, June 1996, pp. 131–136.

[18] V. Chikermane, J. Lee, and J. H. Patel, "Addressing design for testability at the architectural level," *IEEE Trans. Computer-Aided-Design*, vol. 13, pp. 920–934, July 1994.

[19] J. Steensma, F. Catthoor, and H. De Man, "Partial scan at the register-transfer level," in *Proc. Int. Test Conf.*, Sept. 1993, pp. 488–497.

[20] T. Thomas, P. Vishakantaiah, and J. A. Abraham, "Impact of behavioral modifications for testability," in *Proc. VLSI Test Symp.*, Apr. 1994, pp. 427–432.

[21] T.-C. Lee, N. K. Jha, and W. H. Wolf, "Behavioral synthesis of highly testable data paths under nonscan and partial scan environments," in *Proc. 30th Design Automation Conf.*, June 1993, pp. 292–297.

[22] M. Potkonjak, S. Dey, and R. Roy, "Behavioral synthesis of area efficient testable designs using interaction between hardware sharing and partial scan," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1141–1154, Sept. 1995.

[23] S. K. Chiu and C. Papachristou, "A built-in self-testing approach for minimizing hardware overhead," in *Proc. Int. Conf. Computer Design*, Oct. 1991.

[24] L. J. Avra, "Allocation and assignment in high-level synthesis for self-testable data paths," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 463–472.

[25] M. Nourani and C. Papachristo, "Structural BIST insertion using behavioral test analysis," in *Proc. European Design and Test Conf.*, Mar. 1997, pp. 64–68.

[26] S. Bhattacharya and S. Dey, "H-SCAN: A high level alternative to full scan testing with reduced area and test application time," in *Proc. VLSI Test Symp.*, Apr. 1996, pp. 659–668.

[27] R. B. Norwood and E. J. McCluskey, "Low overhead scan data paths," in *Proc. Int. Test Conf.*, Oct. 1996, pp. 659–668.

[28] M. S. Abadir and M. A. Bruer, "A knowledge based system for designing testable VLSI chips," *IEEE Design Test Comput.*, vol. 2, no. 4, pp. 56–68, Aug. 1985.

[29] S. Bhattacharya, S. Dey, and B. Sengupta, "An RTL methodology to enable low overhead combinational testing," in *Proc. European Design and Test Conf.*, Mar. 1997, pp. 146–152.

[30] S. Bhatia and N. K. Jha, "Genesis: A behavioral synthesis system for hierarchical testability," in *Proc. European Test Conf.*, Mar. 1994, pp. 272–276.

[31] I. Ghosh, A. Raghunathan, and N. K. Jha, "A design for testability technique for RTL circuits using control/data flow extraction," in *Proc. Int. Conf. Computer-Aided-Design*, Nov. 1996, pp. 329–336.

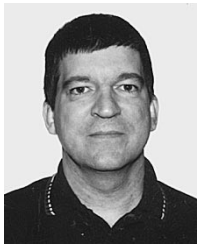
[32] Synopsys User's Manual, Synopsys Inc., Mountain View, CA, 1994.

Samir Boubezari (S'94–M'99) received the B.Sc. degree in electrical engineering from Constantine University, Algeria, in 1989 and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Ecole Polytechnique de Montréal, Montréal, PQ, Canada, in 1993 and 1998, respectively.

Currently, he is a Senior R&D Engineer in the Test Synthesis Group at Synopsys, Inc., Mountain View, CA. His principal research interests include design-for-test, built-in self-test, and high-level testing of digital circuits.

Eduard Cerny (M'73–SM'91), for a photograph and biography, see p. 345 of the March 1999 issue of this TRANSACTIONS.

Bozena Kaminska (M'88), for a photograph and biography, see p. 345 of the March 1999 issue of this TRANSACTIONS.



Benoit Nadeau-Dostie (S'80–M'82–SM'95) received the Ph.D. degree in electrical engineering from Université de Sherbrooke, Sherbrooke, PQ, Canada, in 1985.

He has been a Chief Scientist at LogicVision, Ottawa, Ont., Canada, since 1994. From 1986 to 1994, he was an Advisory Engineer at Bell-Northern Research (BNR). He was the main architect of BNR's Design-for-Testability (DFT) strategy. From 1985 to 1986, he was with the Department of Electrical Engineering of the Université Laval (Vision and Digital Systems Lab). His contributions were an auditory prosthesis based on a microelectronics neural stimulator. He has published several articles and holds five US patents related to memory, logic, and board testing. His interests are in defining test strategies and algorithms with a focus on embedded tests.