A Controller Testability Analysis and Enhancement Technique

Xinli Gu
Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043, USA

Erik Larsson, Krzysztof Kuchinski and Zebo Peng
Dept. of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

This paper presents a testability analysis and improvement technique for the controller of an RT level design. It detects hard-to-reach states by analyzing both the data path and the controller of a design. The controller is modified using register initialization, branch control, and loop termination methods to enhance its state reachability. This technique complements the data path scan method and can be used to avoid scanning registers involved in the critical paths. Experimental results show the improvement of fault coverage with a very low area overhead.

1. Introduction

Many design for testability (DFT) techniques have been developed to ease testing problems. However, they usually require large area overhead and may degrade the performance of a circuit. Some approaches have been proposed to reduce these drawbacks by using techniques which have low area and performance impact, such as partial scan design [1, 2, 3, 4, 5]. For example, a testability analysis and improvement technique based on partial scan is presented in [3]. In this approach, design testability is measured for all nodes in the data path. Although the difficulty of state transitions in the controller is considered in the data path testability analysis, there is no testability measurement for the controller. Partial scan register selection is only carried out for the data path of a design. When the selected registers are involved in critical paths, this approach can not be used.

Recently Dey *et al*. proposed a DFT technique to improve the controller testability for designs which consists of a controller and a data path [8]. A technique has been developed to identify the control signal conflicts due to control signal correlation imposed by the controller specification. The controller is re-designed in such a way that the identified implications are eliminated by adding extra control vectors.

A synthesis-for-testability approach that uses control points at the conditional branches to improve testability has been proposed [9,10]. An analysis of the controllability of branch conditions in the control-data flow graph identifies hard-to-control loops. The controllability of the hard-to-control loops is enhanced by inserting control points at the exit conditions of these loops. Test statements are also added if necessary to allow hard-to-control variables to be directly controllable from existing primary inputs.

In this paper, we propose a general testability analysis and enhancement technique for the controller of a design. It measures the combinational and sequential hardness to reach any state in the controller. The register initialization, branch control and loop termination methods are developed to improve the state reachability of hard-to-reach states. This technique compliments the data path scanning technique and can be used to avoid scanning registers involved in the critical paths.

The rest of the paper is organized as follows. First our design representation and data path testability analysis are presented. The state reachability definition and its algorithm are then given. They are followed by the state reachability improvement methods. Finally, we conclude the paper by experimental results.

2. Preliminaries

In this section, we introduce our design representation and testability analysis of data path. Our design environment allows designers to specify their designs in behavioral VHDL. The specification is later translated into an internal representation, called ETPN [6] which consists of two parts: a data path and a controller. Figure 1 presents an example of a behavioral VHDL specification and the corresponding ETPN representation. The data path is a directed graph with nodes and lines (arcs) where a node represents storage or manipulation of data and a line connecting two nodes represents the flow of data. The controller is modeled as a timed Petri net. The two parts are related through the states (Petri net places) in the controller controlling the data transfers in the data path, and the condition signals in the data path controlling some transition(s) in the controller. As an example, in figure 1 state S_4 in the controller is used to control the data transfer from input port P_1 to register Y in the data path. When S_4 holds a token, this transfer will take place. Condition nodes C_1 and $\overline{C}_1$ in the data path control the transitions from S_3 to

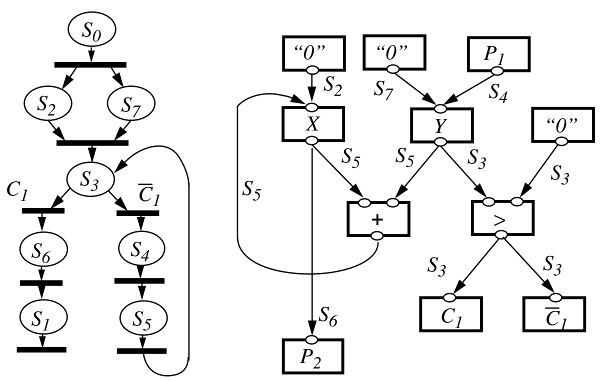
S_6 and the transition from S_3 to S_4 in the controller respectively. State S_0 initially holds a token. The token will be transferred to the other consequent state(s) in the next clock cycle. The execution will terminate when all tokens in the controller are consumed. For example, in figure 1, when the token in state S_1 is consumed the execution will stop.

The testability analysis of the data path [3] is defined by the measurements of controllability and observability. The controllability measures the cost of setting up any specific value on a line. The observability, on the other hand, measures the cost of observing any specific value on a line. The controllability and observability measurements reflect respectively two procedures during test generation and test application, the fault sensitization and the fault propagation. Both the controllability and the observability are further defined by two factors: combinational factor and sequential factor. We have therefore four measurements for testability of the data path: combinational controllability (*CC*), sequential controllability (*SC*), combinational observability (*CO*) and sequential observability (*SO*) [7].

The testability analysis takes into account the structure of a design, the depth from I/O ports and the characteristics of the components used. It reflects the test generation complexity and test application time for achieving high fault coverage. Improving testability in the data path is done by transforming some registers with the worst testability analysis measurements to scan registers [3].

```
ENTITY counter IS
    PORT(P1 : IN INTEGER;
            P2 : OUT INTEGER);
END;
ARCHITECTURE behave OF counter IS
BEGIN
    PROCESS(P1)
        VARIABLE X, Y : INTEGER;
    BEGIN
        X := 0; Y := 0;
        WHILE NOT (Y >0) LOOP
            Y := P1;
            X := X + Y;
        END LOOP;
        P2 <= X;
    END PROCESS;
END;
```

(a) behavioral VHDL



(b) controller (c) data path

Figure 1. A design example in VHDL and ETPN

3. Controller Testability Analysis

The controller testability is measured in terms of the state reachability for each state (Petri net place) of the controller. The state reachability is defined by the difficulty of reaching the state from an initial state. It consists of two measurements: combinational state reachability (CSR_i) and sequential state reachability (SSR_i), for a given state S_i.

Initial State: The initial state, S_0, as illustrated in figure 2(a), has the best state reachability:

$$CSR_0 = 1 \quad (1)$$

$$SSR_0 = 0 \quad (2)$$

CSR_0 is assigned to 1 because the probability of reaching this state is 1 and SSR_0 is assigned to 0 because no clock cycles are required to reach this state.

Simple Construct: A simple construct consists of one transition with a single input place (S_i) and a single output place (S_j) as illustrated in figure 2(b). The state reachability will be calculated as:

$$CSR_j = CSR_i \quad (3)$$

$$SSR_j = SSR_i + 1 \quad (4)$$

The combinational state reachability for state S_j is the same as that of state S_i. The sequential state reachability of S_j is the state reachability of S_i plus one since one more clock cycle is required to reach state S_j.

OR-Construct: An OR-construct consists of a set of transitions such that a state can be reached by any of the transitions in this set. For example, in figure 2(c), state S_k can be reached either by the transition between state S_i and state S_k or by the transition between state S_j and state S_k. The state reachability is calculated based on the assumption that we can always reach state S_k from a state with the best state reachability. Therefore, we have:

$$CSR_k = \begin{cases} CSR_i & \text{if } \left(CSR_i + \dfrac{SSR_L}{SSR_i} > CSR_j + \dfrac{SSR_L}{SSR_j}\right) \\ CSR_j & \text{otherwise.} \end{cases} \quad (5)$$

$$SSR_k = \begin{cases} SSR_i + 1 & \text{if } CSR_k = CSR_i \\ SSR_j + 1 & \text{if } CSR_k = CSR_j \end{cases} \quad (6)$$

where SSR_L is the largest sequential state reachability in the design, which is an estimation of the longest path from the initial state to the terminating state.

Conditional Construct: In a conditional construct, a state can be reached through a transition only if the condition attached to the transition is true. In figure 2(d), state S_i can be reached from state S_k only if condition C is true. Otherwise, $\overline{C}$ is true and state S_j will be reached from state

S_k. The state reachability is calculated by considering the combinational controllability (CC_c) and the sequential controllability (SC_c) of the condition node in the data path:

$$CSR_i = CSR_k \times CC_C \tag{7}$$

$$SSR_i = \text{Max}\{SSR_k, SC_C\} + 1 \tag{8}$$

$$CSR_j = CSR_k \times CC_{\overline{C}} \tag{9}$$

$$SSR_j = \text{Max}\{SSR_k, SC_{\overline{C}}\} + 1 \tag{10}$$

where CC_c is the combinational controllability of the condition attached on the state transition [7]. The SC_c is the sequential controllability of the condition, i.e., the number of clock cycles required to control the condition [7]. If the condition is used to control the exit from a loop which has a very large repetition count, we will have a large SC_c which reflects the implication of this loop construct.

AND-Construct: An AND-construct consists of a transition such that a state is reachable through the transition when all input states to the transition are reached (hold a token). In figure 2(e), S_k is reachable only when both state S_i and state S_j are reached. The state reachability of state S_k is calculated by:

$$CSR_k = CSR_i \times CSR_j \tag{11}$$

$$SSR_k = \text{Max}\{SSR_i, SSR_j\} + 1 \tag{12}$$

Parallel Construct: In a parallel construct, a set of states will be reached by firing of a transition. Figure 2(f). shows a parallel construct. The state reachability of states S_i and S_j in the figure are calculated by the same formula as in the simple construct.

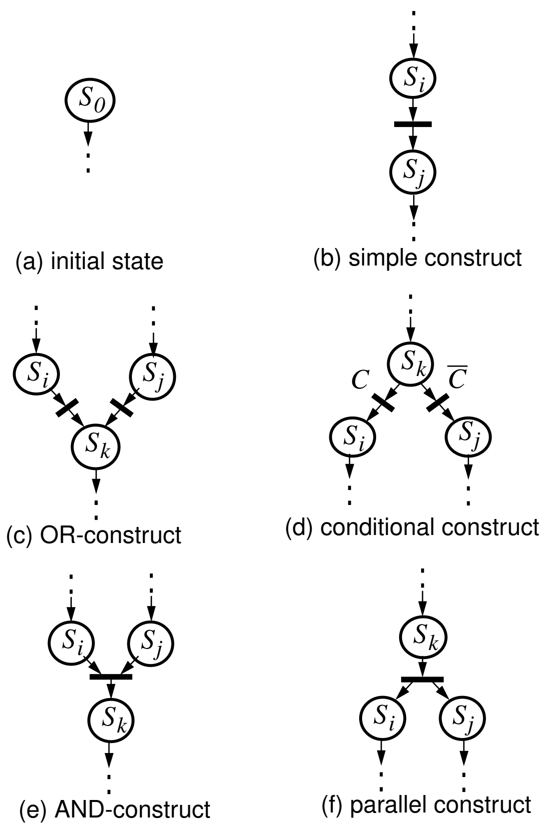


Figure 2. Different constructs

4. State Reachability Analysis Algorithm

The state reachability analysis algorithm calculates the combinational state reachability and the sequential state reachability for all states in a controller. It starts by assigning the state reachability to all initial states and putting these states in a FIFO queue, Q. The breadth-first search strategy is used during the selection of states for calculation. In the next step, one state S is taken out from Q. The construct type of the transition from S to its consequent state(s), for example AND-construct, is checked and the appropriate formulae are used for calculating its CSR and SSR. This procedure is repeated until Q is empty.

A1 Assign all initial states (use formulae 1 and 2).

A2 Put all initial states into queue Q.

A3 Assign the rest of the states with the worst CSR and SSR: CSR := 0; SSR := SSR_L;

A4 If Q is empty, then go to A9;
else assign the first state in Q to S_{prev},
and remove it from Q.

A5 Check the output transition(s) type from S_{prev}:
- a) if it is a simple construct:
  go to A6 (use formulae 3 and 4).
- b) if it is an AND-construct:
  check if all the other input state(s) have been calculated.
  if "yes", go to A6 (use formulae 11 and 12).
  if "no", then put S_{prev} to Q and go to A4.
- c) if it is an OR-construct:
  go to A6 (use formulae 5 and 6).
- d) if it is a conditional construct or parallel construct:
  go to A6 (use formulae 7,8, 9 and 10 or formulae 1 and 2).
- e) if it is a terminating transition (leading to an empty state): go to A4.

A6 Reach the consequent state(s) S_{cons} and calculate its CSR and SSR by the corresponding formulae.

A7 If the newly calculated CSR and SSR are better than stored ones for S_{cons}, replace the stored CSR and SSR by the newly calculated ones and put S_{cons} into Q.

A8 Go to A4.

A9 End.

The calculation of reachability measurements for states included in loops is difficult. Their reachabilities depend not only on some reachabilities already computed but also on reachabilities not yet computed for the states involved in the loop. Our algorithm deals with this problem by first assigning to each state the worst reachability and then updating the reachability only when it is better than the previously assigned value (A7 in the algorithm).

It has to be noted that a loop consists of both conditional and OR-constructs and formulae 5-10 are used to compute their reachabilities. These computations involve calculation of controllability factors for the conditions controlling the loop execution and thus our reachability calculation takes into account the additional difficulty of controlling the loop exit. The controllability factor calculation for conditions is carried out separately during the data path testability analysis process [7].

The algorithm produces two reachability measurements for every state S_i, CSR_i and SSR_i. To evaluate the total state reachability we combine these two measurements using the following formula:

$$CSR_i + k\frac{SSR_L}{SSR_i}$$

where SSR_L is the largest SSR in the design and k is the ratio between CSR and SSR given by designers. This formula is used in selecting the difficult-to-reach states for improvement.

5. Controller Testability Enhancements

After analyzing and evaluating the state reachability for all states in the controller, we can identify the hard-to-reach states. Different techniques are then used to make these states easy to be reached. In the following, we will discuss several of these techniques.

5.1 Register Initialization

When a register in the data path is hard to be initialized due to the hard-to-reach state in the controller, the register initialization/setting technique can be used to improve this situation. Figure 3 illustrates the method of enhancing the controllability of setting/initializing register *Regj* through register *Regi*. This method finds an accessible point in the data path (either a scan register or an input port, such as scan register *Regi* in the figure) which has a short “distance” to the input of the register to be initialized (such as register *Regj* in the figure 1) and a short “distance” from the state controlling the accessible point to the state controlling the register in the controller. The distance in the data path is measured by the number of components between the accessible point and the register. The distance in the controller is measured by the number of transitions between the state controlling the accessible point and the state controlling the register.

In the controller, we improve the state reachability of setting/initializing the register by introducing an extra conditional transition from an initial state to the state controlling the accessible point directly. The condition is controlled by a test signal, *T1*. Thus, the transition can be fired when the *T1* signal is true and we can easily set/initialize a register through the closest accessible point to the register. This method has another more important feature that the start execution point of a circuit can be controlled by transferring token(s) from the initial state(s) directly to the state(s) that we want to start the execution and getting the input value(s) from the input port(s) and/or scan register(s). This feature can significantly improve the efficiency of test generation.

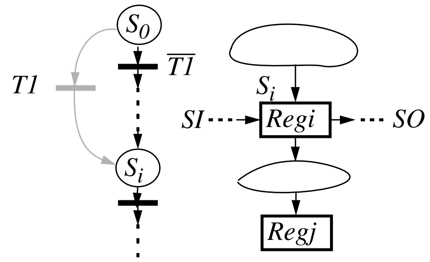


Figure 3. Initialize or set *Regj* through *Regi.*

5.2 Branch Control

The state reachability enhancement for a state which is reached through a transition controlled by a condition is required when the controllability of the condition is poor. We assume that the controllability of condition C in figure 4 is poor. To enhance the state reachability of state S_i, we modify condition C to $C \vee T2$ and $\overline{C}$ to $\overline{C} \wedge \overline{T2}$, where $T2$ is a test signal. When $T2$ is true, the transition controlled by the new condition $C \vee T2$ will be fired, no matter what value C has. If we only need to enhance the reachability of state S_i, i.e., state S_k and other previous states are not required during test, we can use the same method as the control enhancement for register setting/initialization to enhance the reachability of state S_i.

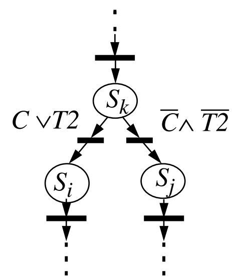


Figure 4. Select branch by *T2*

5.3 Loop Termination

Feedbacks usually take huge computing time in test generation. The control of feedback termination can not only simplify test generation and shorten test application, but more importantly it can increase the fault coverage by making fault detection easier. Assume a loop running from 10 down to 0. The register keeping the loop variable will contain 0 at the end of the loop. It will ease testing if we can get other values in the register at the end of the loop. By adding a test point, we make other values possible. In the example we may terminate the loop at any value from 0 to 10. Thus, we will achieve higher fault coverage.

We assume that the controllability of condition C in figure 5 is poor. To enhance the state reachability of state S_i, we modify condition C to $C \vee T3$ and $\overline{C}$ to $\overline{C} \wedge \overline{T3}$, where $T3$ is a test signal. When $T3$ is true, the transition controlled by the new condition $C \vee T3$ will be fired, no matter what value C has.

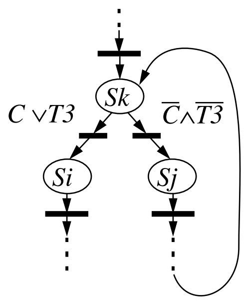


Figure 5. Terminate feedback by *T3*

6. Experiments

We used the Mentor Graphics synthesis and test generation tools as an experimental platform with the default setting used in its test generation process.

The benchmarks we used are a counter, a differential equation (diff), and mag. The results are presented in table 1, where the area is measured in mm^2 . We applied the loop termination technique to two benchmarks, counter and diff. In the counter benchmark, the fault coverage with no DFT technique is 25.23% and the fault coverage with loop termination technique is increased to 84.67%, an improvement of 235.63%. The area overhead is only 0.25%. In general, the loop termination technique has very low area overhead and is efficient when a design has complicated control loop(s). In benchmark mag, there is no loop. We used the branch control technique instead. The fault coverage increased from 51.17% to 65.23% and the area overhead is 0.5%. When the register initialization method is used, considerable improvement in term of fault coverage has also been achieved. For example, with diff, the fault coverage is increased from 13.20% to 98.06%, with a overhead of 1.27%.

Table 1 Summary of Experimental Results

design	DFT technique	fault coverage	area
counter	no DFT	25.23%	0.5525
	loop termination	84.67%	0.5539
diff	no DFT	13.20%	7.3596
	loop termination	96.33%	7.3674
	register initialization	98.06%	7.4534
mag	no DFT	51.17%	1.6435
	branch control	65.23%	1.6528
	register initialization	77.73%	1.6989

7. Conclusions

In this paper, we have presented a method to analyze the testability of a controller. It measures the combinational and sequential hardness to reach each state in the controller. Based on this result, hard-to-reach states are detected and three testability enhancement techniques are developed to improve the state reachability.

This method has the advantage that it does not suffer from the timing penalty which data path scan technique usually does. It can be used as complement to data path scan in order to achieve better test quality and less area and timing penalties.

Experimental results show that this method can effectively improve fault coverage with a very limited area overhead.

8. References

[1] C.-H. Chen, T. Karnik and D. G. Saab, Structural and Behavioral Synthesis for Testability Techniques, *IEEE Trans. Computer-Aided Design*, 13(6), pp. 777-85, 1994.

[2] S. Dey, M. Potkonjak and R. Roy, Exploiting Hardware Sharing in High Level Synthesis for Partial Scan Optimization, *Proc. International Conference on Computer-Aided Design*, November 1993.

[3] X. Gu, K. Kuchcinski and Z. Peng, Testability Analysis and Improvement from VHDL Behavioral Specifications, *Proc. EURO-DAC'94*, pp. 644-649, Grenoble, September 1994.

[4] T. C. Lee, N. K. Jha and W. H. Wolf, Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and Partial Scan Environments, *Proc. Design Automation Conference*, pp. 292-7, 1993.

[5] T. Thomas, P. Vishakantaiah and J. A. Abraham, Impact of Behavioral Modifications for Testability, Proc. the 12th IEEE VLSI Test Symposium, pp. 427-32, April 1994.

[6] Z. Peng and K. Kuchcinski, Automated Transformation of Algorithms into Register-Transfer Level Implementations, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No.2, pp. 150-66, February 1994.

[7] X. Gu, *RT Level Testability Improvement by Testability Analysis and Transformations*, PhD Dissertation, No. 414, Linköping University, Sweden.

[8] S. Dey, V. Gangaram and M Potkonjak, A Controller-Based Design-for-Testability Technique for Controller-Data Path Circuits, *ICCAD 1995*, pp 640-645.

[9] F. Hsu, E. Rudnick, J. Patel, Testability Insertion in Behavioral Descriptions, Proc. *ISSS 1996*, pp 139-144.

[10] F. Hsu, E. Rudnick, J. Patel, Enhancing High-Level Control-Flow for Improved Testability, *Proc. ICCAD 1996*.