Design-for-Testability for Synchronous Sequential Circuits using Locally Available Lines

Irith Pomeranz and Sudhakar M. Reddy +
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242

1. Introduction

We propose a non-scan design-for-testability (*DFT*) method to increase the testability of synchronous sequential circuits. Non-scan *DFT* allows at-speed testing, as opposed to scan or partial-scan based *DFT* that normally leads to low-speed testing and longer test application times due to scan operations. The proposed method is based on the identification of several types of restrictions imposed by the combinational logic of the circuit on the values that can be assigned to the next-state variables. These restrictions limit the set of states the circuit can reach, thus limiting the set of input patterns that can be applied to its combinational logic during normal operation. This in turn limits the fault coverage that can be achieved. The proposed *DFT* procedure is different from other non-scan based *DFT* procedures [1], [2] in that it relies on lines available locally to drive the inserted *DFT* logic, avoiding the routing of primary input lines to the flip-flops, and the routing of internal lines to the primary outputs. The proposed scheme uses the complement value $\bar{Y}$ of a next state variable Y or the value of an adjacent state variable Y' in order to change the value of Y, and thus enrich the set of states that can be reached by the circuit.

The proposed approach considers several special cases that result in unreachable states (or states that cannot be easily reached) to determine where the *DFT* logic will be placed. We consider cases where a next-state variable always (or almost always) carries a single value under a random sequence of input vectors, and cases where two next-state variables carry the same values, or complemented values. These cases have a drastic effect on the set of state variable patterns that can be applied to the combinational logic of the circuit in practical time, thus limiting its testability.

2. *DFT* based on locally available lines

To identify restrictions on and dependencies among values of next state variables, we simulate a random input sequence of fixed length. The circuit is started in the fully-unspecified initial state, and the states it reaches under the selected input sequence are recorded. This does not accurately identify dependencies of the types discussed above. However, a random sequence of input vectors is useful in that it provides information about state variables that are *difficult* to set to specific values, as well as state variables that *cannot* be set to specific values.

Due to the use of the fully-unspecified state as the initial state of the circuit, the dependencies among the state variables must accommodate unspecified values (x). To define these dependencies, we use the following notation. The input sequence applied to the circuit in order to find the dependencies is denoted by V. We use $V[u]$ to denote the input pattern at time unit u of V. We denote by S_u the state of the circuit at time unit u. The value of state variable y_i under S_u is denoted by $S_u[i]$. The dependencies are defined with respect to V and are written in terms of the present state variables (denoted by y or y_i).

Constant {0,1} state variables: State variable y_i is said to be constant α, $\alpha \in \{0,1\}$, if $S_u[i] \neq \bar{\alpha}$ for every time unit u along V, and there exists a time unit u_0 such that $S_{u_0}[i] = \alpha$.

Constant x state variables: State variable y_i is said to be constant x if $S_u[i] = x$ for every time unit u along V.

Equal state variables: Two non-constant state variables y_i and y_j are said to be equal if for every time unit u along V, either $S_u[i] = x$, or $S_u[j] = x$ or $S_u[i] = S_u[j]$.

Complementing state variables: Two non-constant state variables y_i and y_j are said to be complementing if for every time unit u along V, either $S_u[i] = x$, or $S_u[j] = x$ or $S_u[i] \neq S_u[j]$.

We use the following variables to store the dependencies identified among the state variables. $const[i] \in \{0,1,x,-\}$ indicates whether y_i is constant 0, constant 1, constant x, or non-constant, respectively. $same-as[j]$ gives the index of the first state variable y_i such that y_i and y_j are equal by the definition above, and $i \leq j$. $compl[j]$ gives the index of the first state variable y_i such that y_i and y_j are complementing by the definition above, and $i < j$.

To resolve the dependencies defined above, we use the logic in the dashed box of Figure 1, referred to as the *DFT structure*. The *DFT* structure contains a multiplexer with a select input E. When $E = 1$, the value of Y_i produced by the combinational logic of the circuit is transferred to the input of flip-flop i unmodified; when $E = 0$, the complemented value of Y_i produced by the circuit is transferred to the input of flip-flop i. By controlling the value of E, the value of y_i can be controlled.

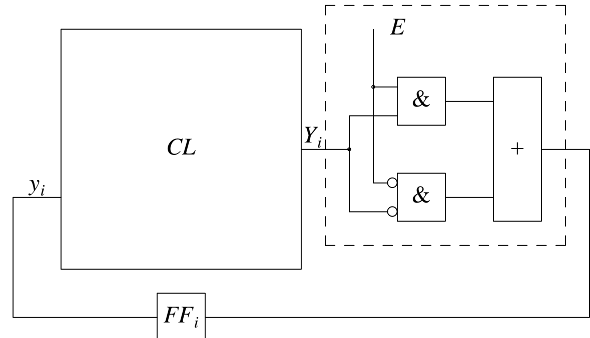

Figure 1: Modification of the next-state logic of Y_i

3. The overall *DFT* procedure

We first add *DFT* structures on the state variables with $const[i] = 0$ or $const[i] = 1$. Suppose that during j iterations, we add *DFT* structures on $Y_1, Y_2, \cdots, Y_j$. Suppose also that before

+ Research supported in part by NSF Grant No. MIP-9357581, and by NSF Grant No. CDA-9601503.

the addition we had $const[1] = const[2] = \cdots = const[i] = 0$ and $const[i+1] = const[i+2] = \cdots = const[j] = 1$. The addition of *DFT* structures in this case allows us to complement $y_1, \cdots, y_j$ simultaneously. As a result of this modification, if the pattern $(y_1 \cdots y_i y_{i+1} \cdots y_j) = (0 \cdots 01 \cdots 1)$ is applicable in the original circuit, then the pattern $(y_1 \cdots y_i y_{i+1} \cdots y_j) = (1 \cdots 10 \cdots 0)$ that cannot be applied in the original circuit is applicable in the modified circuit. Using a single control input E, it is only possible to increase the number of fully specified patterns on $(y_1 y_2 \cdots y_j)$ from one to two. The additional fully specified pattern can be selected in one of several ways. We prefer to place *DFT* structures on all the next state variables with constant values in order to be able to complement all of them.

Next, we consider the state variables with $same-as[i] \neq i$ and the ones with $compl[i] \neq -$. Consider a single pair of state variables $(y_i y_j)$ with $same-as[j] = i$ that can assume the patterns (00) and (11). Placing a *DFT* structure on Y_i or Y_j (but not on both) allows y_i and y_j to be controlled individually, resulting in the patterns {(00),(11)} on $(y_i y_j)$ when $E = 1$ and the patterns {(01),(10)} when $E = 0$. Similarly, if $compl[j] = i$, a *DFT* structure needs to be placed on Y_i or Y_j (but not on both). We use this observation as follows.

We first consider pairs of state variables Y_i, Y_j such that $compl[j] = i$. We select one of the state variables, say Y_i, to place a *DFT* structure on it. We also mark that a *DFT* structure must not be placed on Y_j. Next, we consider state variables $Y_{i_1}, \cdots, Y_{i_k}$ such that $same-as[i_1] = same-as[i_2] = \cdots = same-as[i_k] = i_1$. In this case, we partition the set $\{Y_{i_1}, \cdots, Y_{i_k}\}$ into two (approximately) equal subsets. *DFT* structures are then added only on the variables in one subset.

After each *DFT* structure is placed, we resimulate the input sequence V to update the dependencies among the state variables.

By applying the procedure above again to a circuit that has already been modified, and using a new control input E, it is possible to further divide the sets of equal state variables into smaller subsets, and allow additional state variables that were previously equal to obtain non-equal values. Another consequence of repeating the procedure using a new control input is that state variables that were constant x in the original circuit may be specified once the number of states the circuit can go through is increased.

4. Experimental results

We applied the procedure above to ISCAS-89 benchmark circuits. We used a random sequence of length 200 to collect information about constant, equal and complementing state variables. The results are reported in Table 1 as follows. After circuit name we show the fault coverage achieved for the original circuit by the deterministic test generation procedure of [3]. For the modified circuit, we show the number of state variables modified, and the fault coverage achieved using the test generation procedure of [3]. In this experiment, a single control input is used. It can be seen that significant increases in fault coverage are obtained in many cases. Although the numbers of flip-flops modified by the proposed procedures are higher than those in partial scan designs, the test application time overhead of partial scan is avoided, allowing at-speed testing.

Several methods may be used to increase the fault coverage to 100%. One of them is to use a small number of control inputs instead of only one. To demonstrate the advantages of adding *DFT* structures controlled by multiple inputs, we applied the proposed procedure to s208 and s420 repeatedly until no additional modifications were possible. For s208, the fault coverage achieved by the procedure of [3] reached 100% after four iterations, using four different control inputs. For s420, the fault coverage reached 100% using eight different control inputs.

Table 1: Results of *DFT*

circuit	init f.c	modified ff	modified f.c
s208	69.77	6	82.64
s298	88.64	3	100.00
s382	94.00	13	95.84
s386	81.77	3	90.00
s420	47.44	12	82.39
s526	83.24	13	98.34
s641	87.37	9	100.00
s1423	96.04	38	98.64
s5378	79.14	115	89.19

Table 2: Results of synchronization

circuit	s.v.	init. synch	mod ff	ff [4]
s9234	228	53	51	54
s13207	669	192	129	191
s15850	597	308	82	151
s38417	1636	372	644	643
s38584	1452	1398	28	28

5. Using additional next state variables

In the previous sections, the *DFT* structure placed on state variable Y_i was driven by the functions Y_i and $\bar{Y}_i$. In this section, we consider the possibility of using other next state variables to drive the *DFT* structure of a state variable Y_i. Layout information can be used to restrict the distance allowed between Y_i and Y_j, thus keeping the overhead of routing Y_j to Y_i low. We use this structure to synchronize unsynchronizable circuits. This is possible by using a synchronizable state variable Y_j to drive an unsynchronizable state variable Y_i.

To achieve synchronization, *DFT* structures need to be placed on constant x state variables. We use the dependencies among the remaining state variables to rank them according to the desirability of using them to drive other state variables. We prefer to drive a constant x state variable Y_i from a state variable Y_j such that y_j is not involved in any dependency. We refer to a state variable that satisfies this condition as a *free* state variable, and we denote the set of free state variables by *FREE*. By allowing Y_j to drive Y_i only if $Y_j \in FREE$, we ensure that the *DFT* structure on Y_i does not introduce new dependencies except possibly that y_i and y_j will be equal. We allow non-free state variables with $const[j] \neq x$ to drive other state variables only if the set of free state variables is not sufficient to drive all the state variables with $const[i] = x$. Additional heuristics used are omitted due to space considerations.

We applied the procedure above followed by reverse order simulation to non-synchronizable ISCAS-89 benchmark circuits. The input sequences used were of length 50. We used a single extra input, and added *DFT* structures to synchronize all the state variables. The results are reported in Table 2, as follows. After circuit name we show the number of state variables, and the maximum number of state variables synchronized at any time unit in the original circuit. We then show the number of *DFT* structures placed. For comparison, the number of state variables that needed to be reset using the procedure of [4] to achieve synchronization are shown in the last column of Table 2.

References

[1] S. M. Reddy and R. Dandapani, "Scan design using standard flip-flops," IEEE Design & Test, Feb. 1987, pp. 52-54.

[2] V. Chickermane, E. M. Rudnick, P. Banerjee and J. H. Patel, "Non-Scan Design-for-Testability Techniques for Sequential Circuits", in Proc. 30th Design Autom. Conf., June 1993, pp. 236-241.

[3] X. Lin, I. Pomeranz and S. M. Reddy, "MIX : A Test Generation System for Synchronous Sequential Circuits", in Proc. 1998 VLSI Design Conf., Jan. 1998.

[4] I. Pomeranz and S. M. Reddy, "On the Synchronization of Synchronous Sequential Circuits by Partial Reset using Focused Search", Technical report 9-7-96, ECE Dept., Univ. of Iowa.