# FPGA IMPLEMENTATION OF BIST WITH SELF FEEDBACK LOGIC TO ACHIEVE COMPLETE FAULT COVERAGE

K.Veerraju
M.Tech student,
Pragathi Engineering College,
Surampalem (A.P, IND).

D.Nagesh
*Assitant Professor, Dept.of ECE*
*Pragathi Engineering College,*
Surampalem *(A.P, IND).*

## Abstract

*This paper presents an implementation of a deterministic BIST technique on FPGA that can efficiently achieve complete fault coverage without using any storage devices. A novel test structure containing a self feedback logic unit and a circular shift register is proposed and the self feedback logic unit to generate a set of seed patterns based on responses of some pre selected internal nets .This seed patterns are then used to generate all required test patterns through a circular shift register .It is notice that the self feedback logic unit requires large routing area, when internal nets connects to the self feedback logic. For the reduction of routing area we use arbitrary formulae at the internal connections.*

### 1. Introduction

Logic Built-In Self-Test (BIST) has been shown to be an effective design for testability (DFT) technique in which some on-chip test structure is used to test the digital circuit itself [1]. Pseudo-random testing based on linear feedback shift registers (LFSRs) is commonly used as the basis of BIST due to its simplicity and effectiveness. However a complex circuit often contains some hard-to-detect faults that are random-pattern resistant and thus a pseudo random test scheme usually requires long time to reach satisfactory fault coverage.

To address this problem, in literature several techniques have been proposed. The weighted random test method [1] is presented to enhance the detectability of hard-to-detect faults. This technique may require long test time for a circuit containing many hard-to-detect faults. The mixed-mode BIST technique [1] takes advantages of both pseudo random and deterministic patterns to achieve complete fault coverage in a short time. To carry out the test process, complicated control may be required to switch between different test modes, loading required seeds and/or reconfiguring some specific mapping logic.

In alternative logic BIST methods are proposed that use simple control logic to generate only deterministic patterns without pseudo-random ones. In twisted-ring counters (TRC) along with some reseeding logic are employed to generate the required patterns. Simple finite state diagram based control is used to load seed patterns to the TRC and to perform the TRC operations to generate additional patterns. The required seed patterns can be stored in an on-chip ROM or provided by an external tester. A large number of patterns are generated from a seed pattern so as to reduce the number of required seeds. Though the control is simple, this technique may require long test time to achieve 100% fault coverage if the length of the TRC is large. In [5] a hybrid BIST using partially rotational scan is proposed in which a scan chain is divided into multiple segments, each of which can independently perform rotation operations. The required patterns for complete fault coverage are generated based on a given set of seeds that can be shifted to the scan chain by an external tester and then rotated within each segment. Since only shifting and rotation operations are required, the test control is also simple. However it may still require long test time if a large number of shifting and rotation operations have to be performed.

Some techniques use circuit responses to generate the required deterministic patterns. The circular self-test scheme replaces each primary IO with a special BIST cell and connects them together with the internal scan cells to form a long circular self-test path. Both pattern generation and signature analysis can be done by the self-test path, and thus only small area overhead is needed. However the fault coverage may be degraded if some states required to detect some faults cannot be reached by the self-test path. To address this issue a jumping logic that can change the contents of the scan register to any state is proposed, by which all the desired patterns can be generated efficiently. Authors propose to connect some internal nets of the circuit under test (CUT) to its inputs so as to provide the required logic values of the next pattern directly. By exploring the relation between the responses of internal nets and a pre-computed test set, this method aims to generate a series of deterministic patterns that contain

compatible with the pre-computed ones. However, since the test set is pre-defined, it is difficult to identify a set of connections that can generate all required patterns. To get high enough fault coverage, different sets of connections, each with a specific initial pattern, are required. Therefore the area overhead can be high. All of the above methods need either on-chip ROM or external testers to provide the required seed patterns or some input patterns. In this work we propose a deterministic BIST scheme that requires no storage device. A novel self-feedback logic unit along with a circular shift register design is employed to generate the required patterns. Unlike the work in which selects patterns from a given test set and tries to identify a set of internal nets to provide the required logic values, we develop an efficient method to generate effective test patterns (seeds) based on the current circuit responses. These patterns are then used to regenerate more patterns through a circular shift register. Experimental results show that our method can achieve 100% stuck-at fault coverage using much fewer test cycles than those in [2] and [5]. On average 92.27% and 88.05% test cycle reductions can be achieved compared with [2] and [5], respectively. The comparison also shows that we can use much lower area overhead and comparable test time to reach complete fault coverage. In the rest of this paper we first describe the proposed BIST architecture in Section 2. The method to generate all test patterns and to determine the associated hardware configurations is then detailed in Section 3. Section 4 provides the experimental results along with the comparisons with previous work. Finally, this paper is concluded in Section 5.

## 2 .The Proposed BIST Architecture

As shown in Figure 1 the proposed BIST architecture consists of a self-feedback logic unit (SFL), a circular shift register (CSR), a response monitor and an on-chip BIST control unit. The SFL along with the CSR are used to generate all the required test patterns, and the response monitor is employed to capture the test responses. The whole test procedure is managed by the control unit.

In our BIST architecture all the primary inputs and scan cells of a CUT are serially connected to form the CSR. During test application an initial pattern stored in the CSR is circularly shifted (rotated) by one bit per test cycle. Thus up to $n$-1 additional test patterns can be generated if the length of the CSR is $n$. However it should be pointed out that in our architecture full rotation is not always required for each initial pattern. In other words, each initial pattern can have its own rotation number in order to reach 100% fault coverage in a shorter time. Due to this novel feature, much fewer test cycles are required than previous work [2-5] that uses a fixed number of shift or rotation cycles for each initial pattern or seed.

The required initial patterns for the CSR can be generated by simply resetting the CSR or through the SFL based on the current responses of some pre-determined internal nets. In the latter case, some logic operations of the response bits are employed to change the state of the CSR directly. In this work unary logic operations including no-operation, inversion, short-to-VDD and short-to-GND, and binary ones including and/or, and/nor and xor/xnor are considered. Throughout this paper we shall define each such a logic operation of response bit(s) as one feedback candidate. Some related definitions are also given: a set of feedback candidates that together can provide the required logic values of an initial pattern is defined as a configuration; a pattern generated by the SFL via a configuration is defined as a feedback pattern, the set containing all feedback patterns is called the feedback test set, and a pattern generated by a rotation operation based on a feedback pattern is defined as a rotation pattern. To generate all required feedback patterns we may require multiple configurations. In this work one configuration may be reused to generate multiple feedback patterns so as to reduce the area overhead.

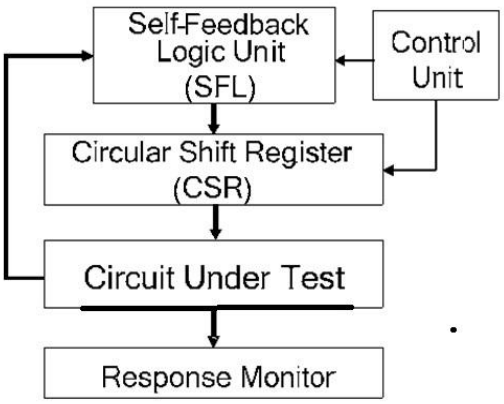



**Figure 1: proposed BIST architecture**

A control unit is employed to control the switching between the self-feedback mode and the circular shifting mode as well as to control the selection of different configurations. Thus the complexity of the control unit depends on 1) the frequency of the switching between the two modes and 2) the number of configurations. Next we present an efficient method which aims to minimize both 1) and 2) described above.

## 3. Test generation and configuration determination

This section presents our method for test generation and configuration determination, which consists of two phases. In Phase 1, a set of feedback patterns are generated with full rotation consideration so as to drop as many easy-to-detect faults as possible. A predefined fault coverage value will be used as a criterion to determine when to stop Phase 1. If any faults are not detected after this phase, Phase 2 is executed in which a new set of feedback patterns are generated with variable rotation number consideration. The main objective of Phase 2 is to determine appropriate rotation numbers for each feedback pattern so as to minimize both test time and test control complexity. Note that in both phases each configuration will be utilized to generate multiple feedback patterns.

**Phase 1: Feedback Pattern Generation with Full Rotation Consideration**

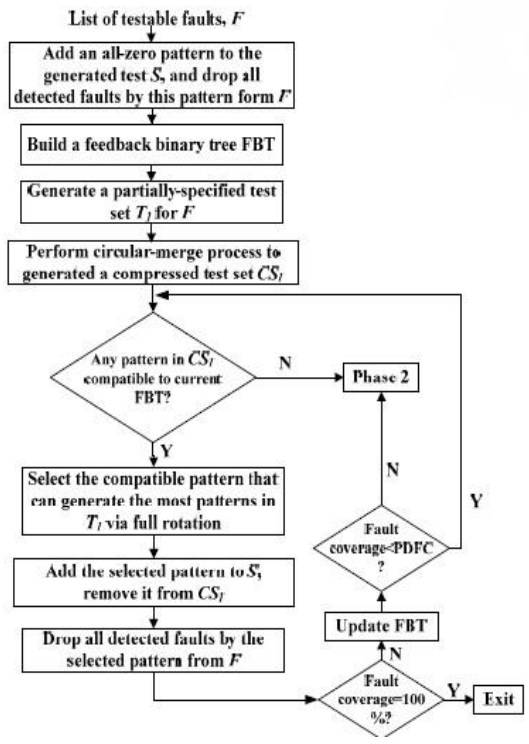



**Figure 2: Procedure of Phase 1**

Figure 2 provides the procedure of Phase 1. At first we use an all-zero pattern as the initial pattern. We add this pattern to the feedback pattern set *S* and drop all faults detected by this pattern from the testable fault list *F*. Based on this pattern, we build a special data structure called a feedback binary tree (FBT) to record the sets of current available feedback candidates that can provide the required logic-0 and logic-1 values for the next feedback pattern. We will use an example shown in Figure 3 to illustrate how a FBT is constructed. Assume the CUT has 4 input pins $\{I_1, I_2, I_3, I_4\}$ and 5 internal nets $\{N_1, N_2, N_3, N_4, N_5\}$. For simplicity, in Figure 3 we directly use internal net responses as feedback candidates without performing any logic operations. After applying the all-zero pattern, the FBT is built in which the root is linked to the all-zero pattern and the two leaf nodes $\{Class_0, Class_1\}$ are linked to the set of feedback candidates that provide logic-0 and logic-1 values, respectively. As indicated, since the responses on the nets of $N_4$ and $N_5$ are logic-0, the $Class_0$ node is linked to the set $\{N_4, N_5\}$. Similarly the $Class_1$ node is linked
to the set $\{N_1, N_2, N_3\}$.

Next a partially-specified test set ($T_1$) is generated for all remaining faults in *F*. This set is used to help determine the feedback patterns to be generated in the following process. We execute a circular merge process on $T_1$ to generate a set of compressed patterns ($CS_1$). An example to generate one compressed pattern is shown in Figure 4, where $P_2$ and $P_3$ can be generated by circularly shifting $P_1$ for one and six cycles, respectively. Thus the three patterns can be merged into one. In our procedure each pattern in $CS_1$ is associated with a weight which indicates the number of patterns that can be generated by this pattern through rotation operations.

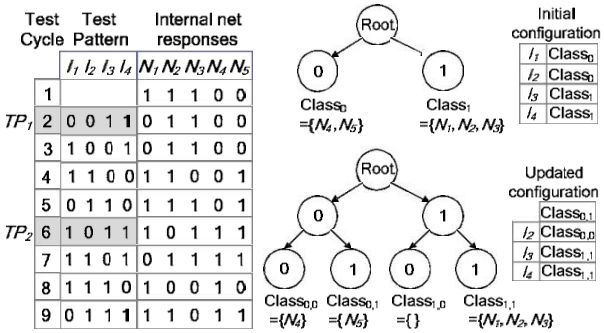



**Figure 3: Example of feedback pattern generation and configuration determination**

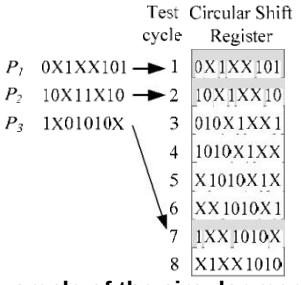



**Figure 4: Example of the circular merge process**

We say that a pattern is compatible to the FBT if the required logic values of the pattern can be provided by the sets of feedback candidates linked to the FBT. Among those patterns in $CS_1$ that are compatible to FBT, we select the one that has the maximum weight as the next feedback pattern. For the example in Figure 3,

assume now the pattern $TP_1$=0011 is such a pattern. This pattern is compatible to FBT because the values of $I_1$ and $I_2$ of $TP_1$ can be provided by any net in {$N_4$, $N_5$} and those of $I_3$ and $I_4$ can be provided by any net in {$N_1$, $N_2$, $N_3$}. This pattern is then used as the first feedback pattern and will be applied at Cycle 2.

Once a proper pattern in $CS_1$ is selected, the pattern is added to $S$ and removed from $CS_1$. Next all the faults detected by the selected pattern and its associated rotation patterns are dropped. If now all testable faults are detected, the procedure ends. Otherwise the FBT will be updated based on the current test responses. Also refer to Figure 3. After fully rotating $TP_1$ we will enter Test Cycle 5 and the responses at this cycle will be used to update the current FBT. Since now the logic value of $N_4$ is 0, $N_4$ is linked to a new leaf node $Class_{0,0}$, which means $N_4$ can provide a logic-0 value for generating $TP_1$ and a Logic-0 value for the next feedback pattern. Similarly $N_5$ and {$N_1$, $N_2$, $N_3$} are linked to new leaf nodes $Class_{0,1}$ and $Class_{1,1}$, respectively, while no net is linked to the node $Class_{1,0}$. After the updating of FBT, if the current fault coverage is equal to or greater than a predefined value (PDFC) then we will enter Phase 2 and the current FBT will be still used. Otherwise another iteration of Phase 1 will be started. In Figure 3, the second feedback pattern $TP_2$ is selected which can be generated by connecting $N_5$ to $I_1$, $N_4$ to $I_2$, and any one net in {$N_1$, $N_2$, $N_3$} to both $I_3$ and $I_4$.

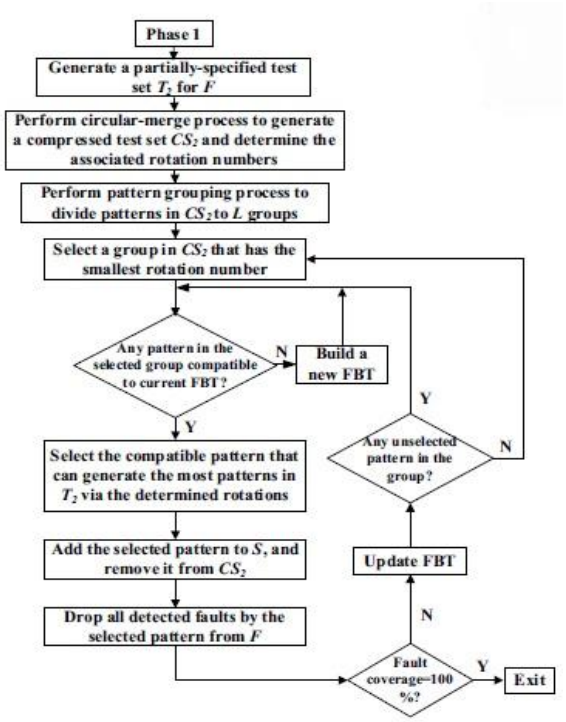



**Figure 5: Procedure of Phase 2**

### Phase 2: Feedback Pattern Generation with Variable Rotation Number Consideration

Figure 5 presents the procedure of Phase 2. Similar to Phase 1, a partially specified test set, $T_2$, for the remaining undetected faults is first generated. A circular merge process is then executed to compress $T_2$. In this process we re-order patterns in $T_2$ in order to determine the smallest rotation number for each pattern in $CS_2$. Refer to Figure 4 again. If the patterns are re-ordered such that $P_3$ is used as the initial pattern, then the other two patterns can be generated using only 3 rotation operations, comparing to the 6 rotations without pattern re-ordering.

To simplify test control, we next divide the patterns in $CS_2$ into $L$ groups such that patterns that have the same or similar rotation numbers will be tentatively put into the same group,and the maximum of the rotation numbers of all patterns in the same group will be used as the rotation number for all patterns in that group. Various tentative grouping results will be evaluated according to their resulting numbers of test cycles, and the one that leads to the fewest test cycles will be adopted.

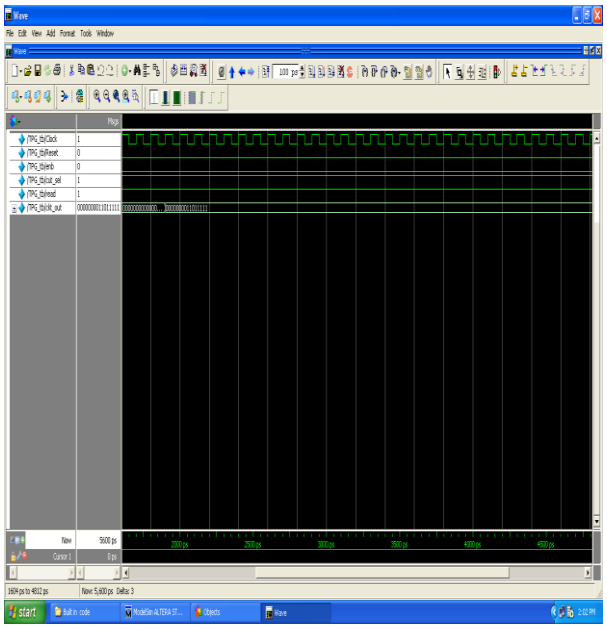
To simplify test control, we next divide the patterns in $CS_2$ into $L$ groups such that patterns that have the same or similar rotation numbers will be tentatively put into the same group, and the maximum of the rotation numbers of all patterns in the same group will be used as the rotation number for all patterns in that group. Various tentative grouping results will be evaluated according to their resulting numbers of test cycles, and the one that leads to the fewest test cycles will be adopted.

We next examine the group that has the smallest rotation number and select a pattern from the group using the same selection criteria as that used in Phase 1: compatible to FBT and can generate the most patterns in $T_2$ via rotation. Once one such pattern is identified, it is added to $S$ and all the faults detected by the pattern and its associated rotation patterns are dropped. If now all testable faults can be detected, the procedure ends. Otherwise additional pattern(s) have to be selected. In this case if no any unselected pattern in the group can be generated using the current configuration, we start a new configuration for which the response of the last pattern of the current configuration is used as its initial pattern and a new FBT is built. Thus our method needs no storage space. On the other hand if all the patterns in the targeted group are already added to $S$, another group of patterns will be processed in the next iteration.

## 4. Experimental Results

Experiments using QUARTUS 12.0 targeting 100% fault coverage. We will compare the results with some related work that also employs only deterministic patterns but needs some space to store the required seed patterns or initial patterns. Quartus II software improves FPGA designer productivity by offering a complete design suite of industry-leading tools and features. The design environment enables FPGAs designers to meet aggressive time-to-market goals with leading-edge synthesis and place-and-route algorithms, as well as advanced DSP design and system integration tools, and includes a broad portfolio of pre-verified IP cores. The development suite targets all aspects of the FPGA design process, from design entry, to timing closure to verificaAltera's Quartus II software improves FPGA designer productivity by offering a complete design suite of industry-leading tools and features.

Here we use 92, 94, 96 and 98% and report the best results. In our method can completely test all circuits using only 1 or 2 configurations and also the number of required feedback patterns (|*S*|) is small.



In the last column of Table 1, we provide the test time in our method and the comparisons with previous work [2] and [5] which use a fixed shift/rotation cycles for each initial pattern or seed. The column "*RD %*" shows the reduction in test time. On average 99.27% and 88.05% test time reduction can be achieved by our method. compares our results with the work in which also generates patterns based on circuit responses. As indicated, in most cases our method can achieve better (100%) fault coverage while using a smaller number of configurations. For some circuits the numbers of test cycles using our method are larger than those reported.

## 5. Conclusions

In this paper we have developed an efficient test-per-clock BIST technique to achieve complete fault coverage and implementing this on FPGA. No storage devices are required by this technique. We employ a self-feedback logic unit to generate a set of seed patterns based on responses of some pre-selected internal nets. The seed patterns are then used to generate all required test patterns through a circular shift register. An efficient method to concurrently determine the test patterns to be generated and the feedback connections is provided which results in much shorter test time or much smaller area overhead compared to previous work.

Although the number of configurations is small, we do notice that the connections from the internal nets to the self-feedback logic unit still require large routing area. We are currently investigating this problem.

## References

[1] L.T. Wang, C.W. Wu, and X. Wen, VLSI Test Principles and Architectures: Design for Testability, Morgan Kaufmann, 2006.

[2] K. Chakrabarty, B. T. Murray and V. Iyengar, "Built-in test generation for high-performance circuits using twisted-ring counters," in Proc. VLSI Test Symp., pp. 22-27, 1999.

[3] S. Swaminathan and K. Chakrabarty, "On using twisted-ring counters for test set embedding in BIST," Journal of Electronic Testing-Theory & Applications 17(6), pp. 529-542, 2001.

[4] B. Zhou, Y.-Z. Ye, Z.-L. Li, J.-W. Zhang, X.-C. Wu and R. Ke, "A test set embedding approach based on twisted-ring counter with few seeds," Integration VLSI Journal 43(1), pp. 81-100, 2010.

[5] K. Ichino, T. Asakawa, S. Fukumoto, K. Iwasaki and S. Kajihara, "Hybrid BIST using partially rotational scan," in Proc. Asian Test Symp., pp. 379-384, 2001.

[6] A. Krasniewski and S. Pilarski, "Circuilar self-test path: A low-cost BIST technique for VLSI circuits," IEEE Trans. on CAD 8(1), pp. 46-55, 1989.

[7] J.Carletta and C.Papachristou, "Structural constraints for circular self-test paths," in Proc. of VLSI Test Symp., 1994, pp. 87-92.

[8] W. Ke, H. Yu and Li Xiaowei, "Deterministic Circular Self Test Path," Tsinghua Science and Tech.12(1), pp. 20-25, 2007.