Self-Test Techniques in SoC for Low Power Consumption

M.Venkateswara Rao¹, N.SaiSravani² and S.P.V.A.Brahmanandum²

¹Asst. Professor Department of ECM, K.L.University, Guntur
²Students, Department of ECM, K.L.University, Guntur
naredla.saisravani8@gmail.com
pavan_sontineni@yahoo.com

Abstract

A generic built-in self-test needed for SoC devices implementing for low power consumption. In this we proposed a new technique to generate a fully pre-computed test set in a deterministic BIST using simple gray counter within a reasonable clock cycles. Conversely, we consider only a small part of the circuit which is to be tested is active and the other parts of the circuit are fed with low leakage input patterns. After that every CUT is fed by a gray counter which makes the overall consumption extremely low. Here we combine this BIST with the external testing strategy for low power consumption using slow-testers. That all concurrent operations are performed in less number of clock pulses. We also consider the SoC devices for the test in the consumption by reducing the number of transitions using a gray counter which stores only one bit during the clock cycle. This approach requires nominal hardware overload over pseudorandom BIST using LFSR or CA.

1. Introduction

A built-in self-test (BIST) or built-in test (BIT) is a mechanism that permits a machine to test itself. Engineers design BISTs to meet requirements such as: High Reliability, low repair clock cycles. Today it is increasingly used for large and complex chips. Based on the application method BIST can be classified in to two types [2]: Test per clock and Test per Scan. In test per clock method, a test pattern is applied to the CUT in every clock cycle and the test response is captured by the test response analyzer. In test per scan scheme, a test pattern is serially loaded into the scan chain and then the pattern is applied to the CUT, thus for an input circuit, n+1 cycles are need to apply a pattern [2]. Based on the pattern generation scheme, BIST can also be classified as pseudo-exhaustive [2], pseudo-random [2][4] and deterministic. Generally, in BIST solution, pseudo-random patterns are generated using LFSR or CA for detecting the random pattern detectable faults. For a random pattern resistant faults is based on deterministic BIST.

In a deterministic BIST, it takes in to account a pre-computed set of patterns and generates the patterns among other patterns. The most trivial way to do that is to store the patterns in a ROM [3] and apply one pattern in every clock cycle. Due to this the area over head is extremely high. The solutions are: using a mapping logic between LFSR and CUT or test set embedding [5].

In a different context power consumption during testing is becoming one of the most important metric of testing as power consumption during testing is much higher than that of normal mode of operation. High average power leads to high heat dissipation, which may damage the chip or package and results in reliability degradation. Peak power causes voltage drop and ground bounce. So, it is mandatory to consider power metric while proposing a test solution. There are many approaches to reduce power consumption in BIST environment. In this paper, we propose a novel technique to embed a fully specified, power aware test pattern set in a BIST environment. The pattern set used is such that it consumes less leakage and dynamic power while applied to CUT. Initially, we tried to generate the whole test set using a single gray counter, that is, for an n input circuit, we assume that an n bit gray counter will generate the whole pattern set. But the cycle length required in this case is more so we have performed a Particle Swarm Optimization (PSO) based column reordering and complement column generation technique over the original low power test set, to reduce the cycle length. Even after this the cycle length to generate the whole test set together is still very high. So, the test set obtained has been broken into smaller subsets and now we try to embed these subsets each one by one and one at a time. The subsets are broken based on the fixed size of the counter. For a particular subset we have to see that the number of inputs changing, is less than or equal to the size of the counter. Remaining bits are all constant.

When all the patterns have been applied, next seed is loaded and the process will continue until all the patterns in the deterministic test set have been generated.

2. Test Pattern Generation

The test pattern generation is same as proposed in [6]. The set thus obtained is rich in don’t cares (test cubes). And these
don't cares are filled to minimize the leakage power using a genetic algorithm based approach [7]. The test set thus obtained in our deterministic test set which we embed in a BIST environment.

3. Particle Swarm Optimization Approach

In this section we perform the pre-processing operations like constant column removal, identical column merging, and complemented column merging, on the low power test set which is obtained from Section II. Basic operations we can perform are column permutation and complemented column generation. These can reduce the cycle length but we have a problem regarding it is NP-hard. PSO can be used to solve this problem for the column permutation and complemented column generation. Here below we described the genetic algorithm which was initially proposed.

A. Genetic Algorithm

Genetic Algorithm represents a class of adaptive search techniques which are mainly employed to solve optimization problems. It includes an initial population which is a set of randomly generated individuals. Each individual is represented by a sequence of variables/parameters (called genes), known as the chromosome. The procedure works, until a stopping criterion is met, as the new populations are generated based on the previous population. The generation of the new population is done through "genetic operators" and the choice of selecting individuals of the current solution that will generate the new population individuals. This algorithm prioritizes the test cases based on the fitness value.

In the genetic algorithm proposed by Li et al. [8], the initial population is fashioned by randomly choosing from the test case pool. The fitness function was calculated as follows:

\[
\text{fitness (pos)} = \text{pos}_i \times \text{pos}_j ; \text{Where pos are position of the test case in the current test suite and n is the population size. The Crossover Algorithm (Recombination) is used to produce two offspring } o_1 \text{ and } o_2 \text{ from two parents } p_1 \text{ and } p_2, \text{ following the ordering chromosome crossover style adopted by Antoniol et al. [9]. A random position } k \text{ is selected in the chromosome.}
\]

The first k elements of p1 become the first k elements of o1.

The last n-k elements of o1 are the sequence of the n-k elements which remain when the k elements selected from p1 are removed from p2; o2 is obtained similarly, composed of the first n-k elements of p2 and the remaining elements of p1 (when the first n-k elements of p2 are removed).

The mutation is generally performed by selecting the two test cases and interchanging their positions in the test case sequence.

B. Particle Swarm Optimization (PSO) [10] is a technique developed by Eberhart and Kennedy in 1995. PSO is initialized with a group of particles with random position and searches for optima by updating their position through generations. Here the particle structure [1]. Where column_permutation will be holding the column numbers in which they should appear and complemented_column tells us whether a column be used in its original form or in complemented form.

The PSO based algorithm starts with an initial position of particles with random velocity. For our case; we start with an arbitrary ordering of columns and also random column type.

The fitness of each particle is calculated in terms of cycle length required to produce the entire test set using an n bit gray counter (n being the number of input). For a binary counter [6], the straight distance of a test set T is \( \max(r_i) - \min(r_i) \), where \( r_i \)'s are the values of the binary patterns of the rows of T. The wraparound distance of T is \( 2^n - \max(r_i, r_j) \), where \( r_{i,a,n} \) and \( r_j \) are consecutive row patterns in sorted order and \( r_{i,a,n} > r_j \). The cyclic distance of T is the minimum of its straight and wraparound distances. For a gray code counter, calculating the straight and wraparound distance requires the knowledge of the pattern sequence generated by it. Given an n bit pattern, we should be able to tell the sequence number in which the pattern will be produced starting from an initial pattern in a gray counter. For simplicity, we consider a reflected gray counter for generation of patterns starting from all 0 patterns. The following pseudo code is used to calculate the sequence number of a pattern in a reflected gray counter [1]. Once the sequence of each pattern is known, it is a trivial work to find the cycle length. The patterns are sorted according to their sequence numbers. Then the same method as described for binary counter is used. The cycle length is used as a fitness for the particles and we are trying to minimize the cycle length in the PSO. The PSO results are shown in Table I [1]. We achieve on an average 52.2% reduction over original cycle length but as it can be seen from column 3; still the pattern cycle length is very high. So in order to this solution we proposed to break the pattern set in to subsets.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Original cycle length</th>
<th>After PSO cycle length</th>
<th>% Reduction</th>
<th>Time ( in ms )</th>
</tr>
</thead>
<tbody>
<tr>
<td>s208</td>
<td>334512</td>
<td>89370</td>
<td>54.0</td>
<td>0.98</td>
</tr>
<tr>
<td>s382</td>
<td>9078783</td>
<td>4194305</td>
<td>52.8</td>
<td>1.54</td>
</tr>
<tr>
<td>s400</td>
<td>61722</td>
<td>43556</td>
<td>30.1</td>
<td>1.22</td>
</tr>
<tr>
<td>s526</td>
<td>8086382</td>
<td>4194326</td>
<td>51.8</td>
<td>11.76</td>
</tr>
<tr>
<td>s713</td>
<td>1.35E+16</td>
<td>4.35E+13</td>
<td>62.7</td>
<td>112.28</td>
</tr>
<tr>
<td>s820</td>
<td>5718607</td>
<td>3097492</td>
<td>54.5</td>
<td>14.13</td>
</tr>
<tr>
<td>s955</td>
<td>1.17E+11</td>
<td>3.44E+10</td>
<td>71.5</td>
<td>22.84</td>
</tr>
<tr>
<td>Avg. Reduction</td>
<td>52.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: PSO Results.
4. Frame Work

For an n-input circuit, we use a k bit gray counter. The rest n-k inputs will be fed directly from memory or from an external low cost, slow tester. For any seed, the n-k input are remaining constant for the entire cycle length of pattern generation. The changing k inputs are also fed using a gray counter. So the overall dynamic power consumption is very less. Another advantage of this approach is that the idle part of CUT is fed by low leakage patterns. The don’t care bits of each pattern are filled to minimize leakage. So, the leakage power consumption of the CUT is also reduced. This is shown in figure 1.

The proposed approach is test per clock scheme where at each clock cycle one pattern is applied. The advantage of this approach is that it can also work with conventional scan design. In fig. 2 [1], we show how it can be added with scan design. But in this case, the seeds have to be stored in the memory. Here, Mux 1 decides whether a constant value or gray counter value will be fed to a scan input. Mux 2 and the D flip-flop constitute the conventional scan D flops. Another advantage of this approach is that the multiplexers placed are not in the functional critical path and hence the maximum operating frequency for functional purpose does not get affected.

The length of a counter depends not only on the nature of the circuit and its test set, but also on number of seeds to be stored in memory and the test cycles. As shown in the experimental result section, there is a good amount of tradeoff existing between counter length, number of seeds and also number of clock cycles required.

5. Algorithm for Finding Subsets

The objective here is to find a subset of the total set \( T_D \) depending on a predefined counter length (size) in such a way that the number of inputs changing in the smaller set should be less or equal to the counter length. Rest of the inputs remain constant for the entire subset. This process continues until we cover all the patterns in test set \( T_D \). The algorithm takes as input the test set \( T_D \) and an integer Counter Length.

Algorithm: Finding sub set (Inputs: \( T_D \) and Counter Length)

**Inputs:** Test Set (\( T_D \)) and Counter Length

**Outputs:** Subsets of \( T_D \) depending on Counter Length

**Step 1:** For each pattern, find how many and what are the bit positions of it that matches with rest of the patterns. We define for each pattern, a variable, no of pattern matching which holds the value of the number of other patterns match that with this pattern with number of bit positions greater than or equal to the specified limit (Number of Input - Counter Length).

**Step 2:** If for each pattern, no of pattern matching is zero, then all these patterns have to be stored in the memory and the algorithm terminates. Otherwise go to step 3.

**Step 3:** Find the pattern with maximum no of pattern matching. Set \( T_{sub} = \text{Null} \). Add the pattern to \( T_{sub} \).

**Step 4:** Find the pattern which matches in the maximum positions. If the number of matches is less than (number of input-counter length) go to Step 6, otherwise go to Step 5.

**Step 5:** Let the pattern be \( t_j \). Add this pattern to \( T_{sub} \), that is, \( T_{sub} = T_{sub} \cup t_j \). Go to Step 4.

**Step 6:** the set \( T_{sub} \) thus obtained holds the patterns which have at least (number of input-counter length) number of bits matching. Write \( T_{sub} \) in a file and set as in [1]. After go to Step 1.

After generating the subsets, a proper seed for each set is selected. The seed selection is determined by the cycle length. Straight and wraparound distances for each set are calculated according to the method described in Section IV. If straight distance is less than wraparound distance, then the pattern with least sequence number is used as seed, otherwise, the initial pattern of wraparound sequence is used as seed. For each file, one seed pattern is required. So the total number of seeds required is equal to the number of files generated plus the patterns obtained in Step 2.

6. Experiment Results

Here in this experiment, we consider a fully specified low power pattern set capable of detecting all the detectable faults in the CUT. The table below shows the test set embedding results. Column 2 notes the counter size which is actually given as an input to the algorithm proposed in Section V. The 3rd column indicates the number of seed patterns required. These patterns can be directly stored in the ROM or a slow speed, low cost tester can be used to feed the patterns. Column 4 indicates the number of initial low power patterns and column 5 notes down the total number of clock cycles required to generate all the low power patterns in our method. Since it is very difficult to directly compare our results with existing techniques, we define a variable named embedding efficiency and compare our results based on that. The embedding efficiency is defined as a ratio of initial number of patterns to the number of patterns required to embed the initial set.
### 7. Conclusion

In this paper, we achieved 85.06% reduction in dynamic power in comparison with pseudorandom testing using LFSR. We also achieved 4.23% reduction in leakage power. We have proposed a novel BIST architecture using a gray counter based approach in such a way that within a moderate clock cycle, a fully specified, low power test set can be embedded. This approach successfully bridges two different aspects of BIST - test set embedding and low power testing. Therefore, the above mentioned algorithm steps would be helpful in minimizing the number of transactions using a gray counter.

As the gray counter uses only one bit to store the result there by helpful in reducing the total power that is used and the power that is dissipated for the system on chip devices using the number of clock cycles and length of the counter taken as input. Another work still left is to find a proper choice of test length. We are trying to formulate an algorithm for finding the proper counter length depending on the number of seeds and embedding cycle length.

The hardware overhead of the approach is not high. The only overhead is storing the seed patterns and that of the multiplexers.

### References


