Embedded Test Control Schemes for Compression in SOCs

Douglas Kay and Sung Chung
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134Address
408 525 8409
dkay@cisco.com, schung@cisco.com,

Samiha Mourad
Santa Clara University
500 El Camino Real
Santa Clara, CA 95053
408 554 4163
smourad@scu.edu

ABSTRACT

This paper presents novel control schemes for testing embedded cores in a system on a chip (SOC). It converts a traditional BIST scheme into an externally controllable scheme to achieve a high test quality within optimal test execution time without inserting test points. The scheme promotes design and test reuse without revealing IP information.

Categories

Testing; Test generation and debugging

General Terms

Algorithms, Performance, Design, Experimentation

Key words:

BIST, Data Compression, Test Resource Allocation

1. INTRODUCTION

Test costs represent a major and rapidly increasing percentage of IC's manufacturing costs. Also, as the technology features became smaller, the number of gates per chip has increased. Although the I/O pins count increased as well, the gate to pin ratio has decreased. As a consequence, the test data bandwidth is becoming extremely large. Testing is further complicated by the fact that most ICs are in effect whole systems on a chip (SOC). Each SOC consists of several cores, thus accessing the cores within the chip is another challenging problem to tackle. To alleviate some of these problems, embedded test is presently favored.

Built-In Self-Test (BIST) is a promising approach for testing embedded core-based systems because of its flexibility in adapting a single LFSR to generate patterns for different cores [3], [4]. Furthermore, BIST makes chip testing less dependent on the external tester and reduces the test data volume and bandwidth between the chip and the tester. The effectiveness of BIST is dependent on the fault coverage achieved, the hardware overhead, and the test execution time. While pseudorandom test pattern generation (PRTPG) costs lower than deterministic test pattern generation, its limitations need to be addressed to improve their effectiveness in testing.

LFSR generated pseudo-random (PR) test patterns are usually longer than deterministic tests to achieve comparable fault coverage. In addition, because of the linear dependency and auto-correlation of these patterns, they are not as effective in detecting faults. Although, in general, PR test sets may detect up to 80% of the faults within reasonable test application time, some faults require test patterns that have a low probability of being generated. These faults are known as *random pattern resistant* faults (RPR). Complete fault coverage may be achieved by steering the LFSRs toward increasing the probability of generating test patterns for these RPR faults. They include: 1) re-seeding techniques [11], [5], [19], [16], 2) weighted random pattern [12], [13], and 3) methods for altering the output sequence of LFSRs [15],[17]. For the third approach, the sequence needed to alter the test patterns is generated on demand by some circuitry. Our method is to generate the sequence in a programmable way and apply it interleaved during BIST operation [8, 9].

In a deterministic BIST environment, the method of altering the test patterns to detect RPR faults requires that the control data be generated by an on-chip circuit. In this paper, we concentrate on a methodology that minimizes test data volume by compressing the data set that controls the PRPG. Compression technique in testing, in the past, has been applied to the raw test patterns to reduce test data volume in the tester. This in turn reduces the data download time from the workstation to the tester. [18], [6]. Recently, it has been used for the full scan test patterns, which will be decompressed on the chip. [7] [1, 2]. Different compression schemes have been used for this purpose. In our case, we applied our special coding [10] to the control data to achieve better compression effectiveness as compared with the one applied to the original raw test data. The decompression is performed on the chip during BIST. We call our test scheme as interactive BIST (iBIST). One of our possible general schemes is outlined in Fig. 1.

To minimize the size of U, we need to minimize the cycles of controlling the LFSR and to carefully apply the appropriate bit values. In this paper, for a given code, we examine different schemes for controlling the LFSR on the overall compaction effects of the data. We formulate the problem in Section 2. In Section 3, we define the control data attributes that are used in comparing the various LFSR control schemes described in Section 4. In Section 5, actual BIST controller implementation and its area overhead will be discussed. And in the last section, we conclude with a summary. First, however, we formulate the problem in the next section.

2. PROBLEM FORMULATION

Since a standard LFSR is used in BIST as a PRTPG, it may seem strange to ask the question whether producing desired test patterns, at will, would be possible. If it is possible, this will lead to generating tests for faults other than stuck-at-fault. Figure 2 shows the situation where the desired set of target test pattern is given for a scan testing. Consider the case where the LFSR should provide the desired patterns $\boldsymbol{t_t}$ = (0 x 0 x 1) starting from its initial state value **S**i = (0 0 1).

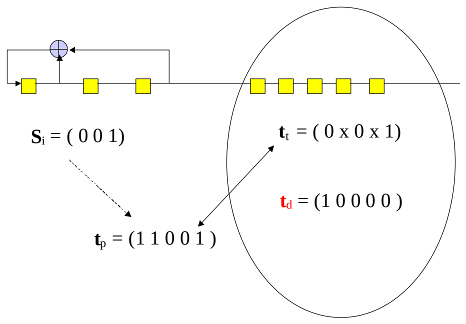
Given a sequential circuit C:

1. Generate a deterministic test D for C and let its length be L
2. Generate a pseudo-random test R for C using a primitive polynomial LFSR
3. Compare test sets D and R and replace unspecified bits, usually indicated by x (don't care), to match the D patterns with R patterns. The subset of unmatched patterns in D represents the patterns needed for RPR faults.
4. Apply a signal set U that controls the LFSR to generate the patterns for RPR faults. Let Lu be the length of U.
5. Compress U to Uc with length Lc and store (transmit) Uc on (to) the Chip.
6. At testing time, decompress Uc to U on chip and apply U to the LFSR.

Figure 1 The General Testing Scheme

In this configuration, LFSR will produce the test vector, $\mathbf{t}_p$ = (1 1 0 0 1) after five shift clocks and arrive at the final state $\mathbf{S}_f$ = (1 0 1) via intermediate states highlighted in the set of contiguous LFSR states, **S** = {(0 0 1), **(1 0 0), (1 1 0), (1 1 1), (0 1 1)**, (1 0 1)}. We can tell the last bit of each state will be shifted into the scan chain to produce $\mathbf{t}_p$. When $\mathbf{t}_p$ is compared with $\mathbf{t}_t$, the difference vector can be derived from the equation $\mathbf{t}_d = \mathbf{t}_t \oplus \mathbf{t}_p$ = {1 0 0 0 0}.

Adding control tap (a 2-input XOR gate) into the different locations of LFSR to flip the output value of the flip-flop located in front of the control tap can make $\mathbf{t}_d$ become all zero vector. Where should one insert the control tap? There are different positions within the LFSR, as we will mention later. For now, we place it right at the output of the LFSR as shown in Fig. 3: the control vector, **U** = (0 0 0 0 1), transforms $\mathbf{t}_d$ to a null vector. The vector is applied from left to right in the time domain. Next, we can think of other positions anywhere inside LFSR.



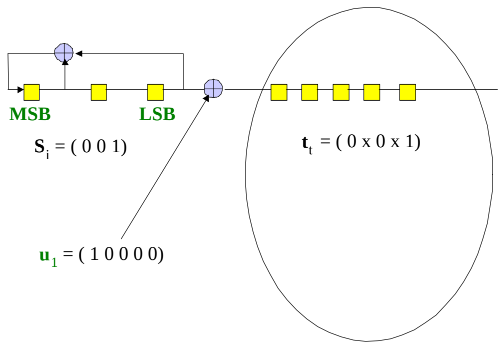
$\mathbf{S}_i$ = (0 0 1)
$\mathbf{t}_t$ = (0 x 0 x 1)
$\mathbf{t}_d$ = (1 0 0 0 0)
$\mathbf{t}_p$ = (1 1 0 0 1)

Figure 2 Generic Scan-BIST

As you move the tap position to the left, you will find out that you need to begin the control vector application as many clock cycles early as the number of your move from the beginning position, i.e. output of least significant flip-flop. As long as you apply the control vectors with the proper timing, you are able to produce desired patterns at will. To describe the problem formally, we define some terminology:

__Definition__: An LFSR consists of several flip-flops, say *N*. The LFSR states are the binary values extracted from its flip-flops. We partition these states into two types: S0 and S1. S0 (S1) states are such that their least significant bit (LSB) is 0 (1).

__Theorem__: Assume the flip-flops are labeled in the descending order from left to right, *N* to 1. Then, the control input u(n) applied at nth clock cycle of any tap located in front of k-th flip-flop, where $0 < k < N + 1$, can steer the LFSR state, S, into S0 or S1 at (n+k)th clock cycles.



MSB
LSB
$\mathbf{S}_i$ = (0 0 1)
$\mathbf{t}_t$ = (0 x 0 x 1)
$\mathbf{u}_1$ = (1 0 0 0 0)

Figure 3 iBIST: Controlling the test pattern stream

Proof: If the tap is located at the front of Kth flip-flop, the input value, $\mathbf{I}_k$ of k-th flip-flop, at a certain discrete time point n , can be set to either 0 or 1 by supplying the input u(n) such that $\mathbf{I}_k(n) = R_{k+1}(n) \oplus u(n)$, where $R_{k+1}(n) \in (0, 1)$, $u(n) \in (0,1)$, and n = 0, 1, 2, …, L, where L is any positive integer. The input value, $\mathbf{I}_k(n)$, can be propagated to the flip-flop R_1 in k clock cycles because $\mathbf{I}_k(n) = \mathbf{I}_{k-1}(n+1) = \ldots = \mathbf{I}_1(n+k-1) = R_1(n+k)$.

Since R_1 (n+k) can be set to 0 or 1, the LFSR state belongs to S0 or S1 at the (n+k)th discrete time.

__Corollary.__ The control input can be calculated as the following equation.

$U(n) = R_{k+1}(n) \oplus \mathbf{I}_k(n) = R_{k+1}(n) \oplus R_1(n+k)$

We targeted only pattern $\mathbf{t}_t$ so far for the sake of explanation. In the case of a whole test of m vectors, $\boldsymbol{T_t}$ = {$\mathbf{t}_1$, $\mathbf{t}_2$, …, $\mathbf{t}_m$}, we repeat the process for each pattern **ti** and develop *m* vectors each of dimension *n*. All such vectors form the control vector **U.** Therefore, the problem that we are solving can be stated concisely as follows.

__The problem__: To find the vector **U** that maps the PR test set, Tp into the deterministic test Ti, $\boldsymbol{f}(\mathbf{U})$: $\mathbf{T}_p \Rightarrow \mathbf{T}_t$ such that the size of **U** is minimal.

The above problem statement can be rephrased as: Find the vector **U** that transforms a projected test set $\mathbf{T}_p$ into target set $\mathbf{T}_t$ while optimizing the associated cost function. The cost function may be the overall size of ***U***, its energy level, or the potential compression gain which would be obtained from using a particular encoding scheme. When we think about a single scan chain, the tap position in Fig. 3 is the easiest because we do not need to worry about the lead time for the start of control data application. However, in reality, the circuits usually contain multiple scan chains and the LFSR drives those multiple scan chains from each output of flip flops through spread network (in IBM terminology) or Phase Shifters (in Mentor Graphics terminology). Therefore, putting a control tap in front of LFSR, shown in Fig. 4, will provide the capability of controlling all of the scan chains without adding multiple control taps in front of each scan chain. However, we can put the control tap anywhere in the LFSR, for experimentation, without loss of generality.

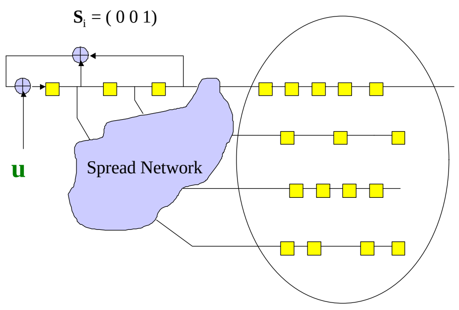


Figure 4. Ideal control tap positions

The control of LFSR may be applied for every pattern of the deterministic test, or if this test is too long, for a selection of the patterns. For example, it is possible to target patterns that are at a certain interval from each other. We refer to this distance as the control interval and denote it by *CI*.

Before describing the LFSR control algorithm, we will define some data attributes that will be used in the formulation and the evaluation of these algorithms.

3. ATTRIBUTES OF THE CONTROL DATA

Data compression consists of mapping a group of bits in the stream, i.e., a source word of length n_i into a code word of length m_i, where m_i is desired to be less than n_i and we define the difference $n_i - m_i$ as the gain g_i, for i = 1, 2,..*N*, where *N* is the total number of words in the source data stream. The size n_i and m_i do not have to be constant for the whole data stream, implying variable to variable coding. The *gain* of the compression scheme should be positive for the majority of words; otherwise, the scheme is not effective in compressing the data. The result of the compression is largely dependent on the characteristics of raw data and the code used to encode it.

Consider the binary stream of test data, length *L*, consisting of runs of 0s and 1s and let these different *runs* be of lengths L_i, with $1 \le i \le N$, then $L = \sum_{i=1}^{i=N} L_i$. To characterize the data and the code, we use the following attributes: *energy*, *sparsity*, and *statistical polarization*.

A test data stream may include an ‘x’ (don’t care) in addition to ‘0’ and ‘1’. We denote the energy by ε and it is defined as the fraction of the 1s in the data stream; the fraction of 0s is denoted by $\bar{e}$. But, since this alphabet of the data may include xs (don't care), the fraction of xs represents the sparsity, s. This is the fraction of the data that can be adjusted to increase or decrease the energy for the data. These three attributes are thus related by the expression: $e + \bar{e} + s = 1$.

The x bits can be replaced with 0 or 1 in a random or algorithmic manner. The data stream can be decomposed into the words, the succession of 0s and 1s. However, if the energy is low, then the majority of words will be succession of 0s in different length by assuming x is converted into 0. Then, instead of looking at the words of the data stream to be the both successions of 0s or 1s, we can define the words to be the successions of 0s, ending with single 1 without loss of generality. This is a conversion into uni-phase source words. These observations led us to develop a code that is suitable for the test data [10]. That is, a code yielding a good compression gain for the type of data. However, we are more interested in finding the overall compression effect of the coding scheme on the source data. That is, we would like to assess the percentage of data reduction after compression to the original deterministic test. We will define the compression effectiveness, *CE*, as : $CE = \frac{L - L_C}{L} 100\%$ where *L* is the length of the deterministic test and Lc that of the compressed data (compressed ***U***). From this definition, we realize that the higher *CE* is, the better the compression scheme. The compression process consists of two step process, preprocessing (steps 1-4 of Fig. 1), and compression (step 5 of the same figure). We can express *compression effectiveness* in terms of the compression through the two processes in the form $CE = 1 - \frac{L_C}{L_u} \frac{L_u}{L} 100\%$.

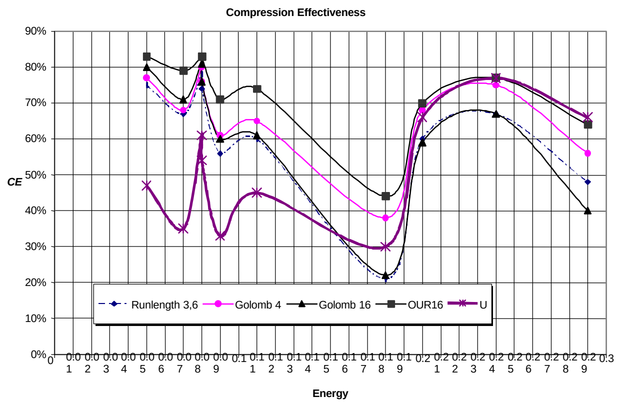


Figure 5. Compression Effectiveness versus Energy.

The compression gain is defined as G =(L_u – L_c). This gain can also be evaluated for each pattern and in this case the Ls refer to the length of the pattern before and after compression. We will use the compression effectiveness to compare the different algorithms presented in the next section.

The code used in this study has been proven to be effective compared to Golomb and Runlength codes. The compression effectiveness for the different codes are plotted against the energy level of the control vectors in Fig. 5. The heavy line represents the *CE* for the vector ***U***. Useful results correspond to points above this line. Among the results, OUR coding was the best. However, we realize that the scheme used to control the LFSR may be further improved over the results obtained. In the next section we present then several algorithms for controlling the LFSR and analyze their efficiency.

4. ALGORITHMS

In the theorem of Section 2, we showed that the LFSR could be controlled to produce a target pattern from any initial state. Controlling the LFSR for a particular target pattern of the deterministic test, changes the state of LFSR and thus affects the control of the remaining patterns. Determining the best control vector requires an exhaustive search. It is an NP complete problem. To make the problem manageable, we developed several efficient algorithms and compared them. The algorithms are all based on a greedy strategy but use different cost functions. The cost function may be (1) the Hamming distance between the target pattern and the projected pattern, (2) the energy level of control vector, ε, (3) the compression gain of encoded control vector, $(L_u - L_c)$.

4.1 Minimum Hamming distance

This algorithm applies to the configuration of Fig. 3. It generates the control vector **u** of the control tap located at the output of the LFSR, the output of its least significant flip-flop. Since the application of control vector does not modify the natural sequence of LFSR states, the control vector **u** is equivalent to the difference vector $\mathbf{t}_d = \mathbf{t}_t \oplus \mathbf{t}_p$, where $\mathbf{t}_t$ is the target pattern from the deterministic test and $\mathbf{t}_p$ is the PR pattern generated by the LFSR. The energy of the $\mathbf{t}_d$ is the Hamming distance between the two patterns.

<u>Minimum Hamming distance algorithm</u>

```
[A greedy algorithm that creates a control data set U with minimal energy level starting with LFSR state si with control interval CI]
1. Td <= D – R ∩D ; [set target pattern set as the remaining patterns after running pseudorandom testing. The size of matrix $T_d$ is m x n.  R is the pseudo random pattern. D is the original deterministic pattern]
2. [Initialization]
   2.1  U <= ∅;
   2.2  s <= si; [LFSR initial state]
   2.3  k <= m;
   2.4  i <= 0;
   2.5  D <= n; [the maximum distance possible for a given pattern]
   2.6  T <= Td;
3. While T ≠ ∅ do
   if ( i / CI)  ≠ 0, then
         s[i]=next(LFSR(s)), tp[i]=out(LFSR(s)) for i = 1 to n; [shift LFSR n times, update s, and generate tp]
         T <= T – {tp} and k <= k -1 if tp ∈ T;
   otherwise
         si = s; [ Save the current LFSR state]
         for j = 1 to k [Try for remaining target patterns]
               s[i]=next(LFSR(s)), tp[i]=out(LFSR(s)) for i = 1 to n; [shift LFSR n times, update s, and generate tp]
               tx = tj ⊕ tp; [Hamming operation];
               if tx < D then
                     (a)  D <= tx;
                     (b)  t <= tj;
                     (c)  Ssave=s;
               end-if
               s <= si [reset the state]
         end_for
         T <= T – {t}; U = U ∪ {u}; [u is derived for target pattern t using LFSR with its initial state s using the corollary in section 2]
         s=Ssavei
     end_if
   i <= i + 1;
end_while
4. Output U
```

4.2 Minimum energy of control vector

If the control tap is located inside the LFSR as indicated in Fig. 6, the applied control vector modifies the LFSR state sequence. This prevents us from using the Hamming distance between the target and projected patterns because the projected pattern will change as every control vector is applied.

We solve the problem by generating the control vector for each pattern of the remaining test set. We Calculate the energy level of all control vectors. Then, we choose the one with the minimum energy. Once the control vector is selected, we remove the target pattern from the remaining test set. The process is repeated until the test set is empty.

4.3 Direct compression gain of control vector

This algorithm is the same as the previous one except that the cost function use is the compression gain, CG, that was defined in Section 3. The calculation of compression gain depends on which type of encoding scheme it uses. In this paper, we use the OUR Coding [Author 2001].

4.4 Progressive Seeding

So far, for the above three algorithms, the control bit is assumed to be a zero whenever the corresponding target pattern bit is a "don't-care". From corollary in Section 2, $U(n) = R_{k+1}(n) \oplus \mathbf{I}_k(n) = R_{k+1}(n) \oplus R_1(n+k)$. $R_1(n+k)$ corresponds to the specific bit of the target pattern at timestamp n + k. If the bit in the target pattern is "don't care", U(n) was automatically set to 0 to lower the control vector's energy. If instead, we allow the choice of setting the control bit to be either 0 or 1, then we have to decide, each time we encounter a don't-care whether to set it to 0 or 1.

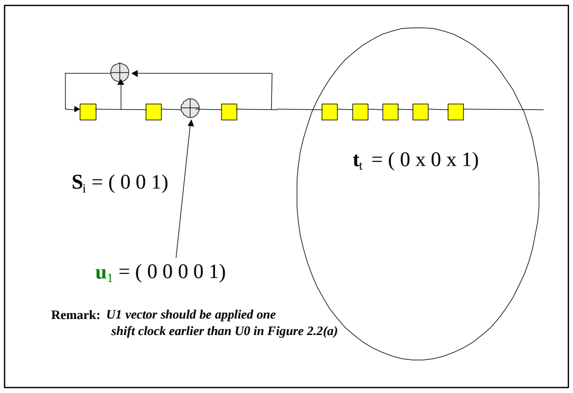

$\mathbf{t}_t$ = (0 x 0 x 1)
$\mathbf{S}_i$ = (0 0 1)
$\mathbf{u}_1$ = (0 0 0 0 1)
Remark: *U1 vector should be applied one shift clock earlier than U0 in Figure 2.2(a)*

Figure 6 Minimum energy control

This choice allows the control of the LFSR, the activity of the ***U*** vector in a progressive manner, on the bit-level instead of the pattern-level. The scheme will provide more granularity of controlling LFSR although it may be more computationally complex. The cost function used for this algorithm continue to be the compression gain, CG.

4.5 Polarity inversion and static value control

We have shown previously, that the compression effectiveness is higher the lower is the energy of the data compressed [10]. Also, in section 3, we showed that the data attributes satisfy the equation $\varepsilon + \bar{e} + \sigma = 1$. Thus when ε, is high, then the inverted data will have low energy. Since we cannot invert the logic for the original pattern, this would be possible if we invert the output of LFSR, instead.

Whether to invert the logic at the output of LFSR can be achieved by adding an extra bit at the first location of each control pattern. This bit should not be counted as part of control vector.

Rather, it is used as a flag bit to set or reset the logic inversion by BIST controller.

Also, some RPR patterns consist of mostly series of 1's or 0's. For these cases, we can supply static high or low input by bypassing LFSR with control gates show in Fig 7. This extra hardware is really very minute with respect to any modern circuit.

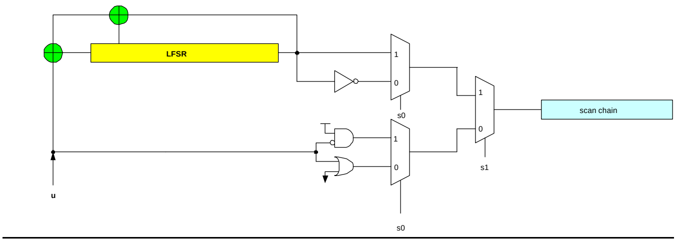


Figure 7 Polarity inversion and static control logic

Assigning {s0, s1} value to the first and the second bit of control vector {ui0, ui1} for ith control pattern, the BIST controller can multiplex the proper logic signal to the target scan chain at the beginning of applying ith control pattern signals.

4.6 Results comparison

iBIST provides the benefit of having compression effect on the original test pattern **D** in two ways. One way is to screen easy to hit test patterns with pseudorandom testing and the other is to apply the control vector to LFSR to hit the remaining or other types of deterministic test patterns. Screening the easy to detect patterns is performed in the same manner by running the same pseudorandom testing for all different control algorithms.

The difference from the traditional BIST fault grading is that we did not use the fault simulation, but used the method of matching between the random pattern, R, and the deterministic pattern, D. We applied coding scheme to the control pattern to compress the control pattern further.

Summing up, we see compression effectiveness in two phases: (1) During pseudorandom testing by filtering easy to detect test pattern, (2) During actual encoding on the control pattern. We applied the different control schemes to several benchmark circuits [14] whose profiles are listed in Table 1.

For each benchmark we calculated the corresponding overall compression effectiveness, CE, of the vector U and plotted the results versus the energy, ε, of the original deterministic pattern. The plots are shown in Fig. 8. As defined in Section 3, the higher the value of CE, the more efficient is the compression scheme.

The polarity inversion and static algorithm yields the best results for 8 out of the 10 circuits. For the other two circuits, it is the progressive seeding algorithm that gave the best results. Also, we would like to draw your attention that most of the algorithm did not perform well on s838 circuit. However, the polarity inversion and the static control scheme brought CE level up to above 70%. Overall, we conclude that the algorithm of Section 4.5, polarity inversion and the static control, yields the best result algorithm

Table 1 Profiles of the Benchmark Circuits

Design	In	Out	scan length	D-test length	PR-test length	PR-fault Cov.	U-test length
ALU	48	18	57	71	10K	66.2	24
Firewire	18	166	258	223	16K	34.5	14
IIR	35	10	243	814	10K	45.0	448
MMX	155	140	298	1883	50K	32.9	1264
Router	70	67	263	576	41.2K	54.3	263
S838	43	3	67	141	30.1K	29.8	99
S1238	22	15	32	205	50K	76.6	48
S13207	39	122	677	231	50K	46.8	123
Sdram	101	86	241	174	18.2K	60.9	68
USB	22	51	118	61	10K	65.6	21

5. PERFORMANCE OVERHEAD OF IBIST

Although BIST testing has been adopted in by the test community, its general acceptance by the design community was quite limited due to several concerns: (1) area overhead, (2) performance degradation during functional operation, and (3) negative impact on project schedule. As the technology feature size plunged to the submicron range, the concern about the area overhead has been eased. Allocating 10 to 20 K gates to the BIST logic became a norm these days. The impact on the performance along the critical path is still challenging because of the need for test point insertion. Our iBIST scheme avoids test point insertion

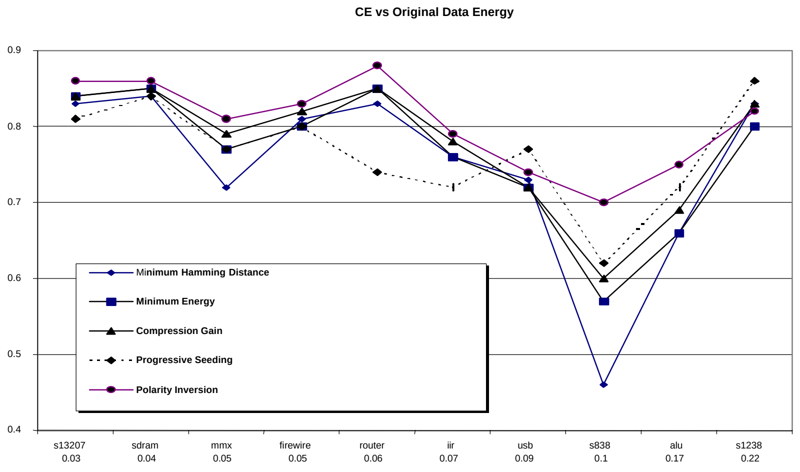


Figure 8 Compression effectiveness

This paper proposes the new BIST method of not using test point insertion, but of using control vectors to target the hard to detect fault in a programmable way. Therefore, the impact on the functional performance is expected no more than the ordinary internal scan implementation. The area overhead and the speed during test mode are listed in Table 2.

The iBIST configuration has been compared to the ordinary BIST implementation. This configuration consists of 5 major parts: (1) BIST controller, (2) LFSR, (3) MISR, (4) shift counter, and (5) pattern counter. The difference between BIST and iBIST design is in the LFSR and the pattern counter module due to their size differences. We used 48 bit for the LFSR and used 24 bit for pattern counter in ordinary BIST implementation. In iBIST implementation case, we could use 16-bit sizes for both LFSR and pattern counter, still satisfying full fault coverage. The synthesis of the designs has been done using Synopsys DC compiler and LSI 0.35 u process library (lcbg10) with worst case set up. We used the 50MHz clock speed, maximal signal transition on the IOs with medium map effort. Both BIST and iBIST could meet the 20 ns clock cycle constraint (for 50 MHz operation) effortlessly with no problem. The major slow down is in the shift and pattern counter. Speeding up these parts, we can run the test in much faster frequency. Using the assumption on the size of LFSR and the pattern counter, iBIST controller area actually came out smaller than the ordinary BIST even after including the decoder module.

Table 2 Comparing iBIST to traditional BIST and controllers

IBST			iBIST	
Module	Area	Speed	Area	Speed
BIST_control	201.32	2.16ns	201.32	2.16ns
LFSR	1666.83	1.89ns	**553.38**	1.85ns
MISR	527.35	1.95ns	527.35	1.95ns
Shift Counter	900.60	6.32ns	900.60	6.32ns
Pattern Cnter	1572.50	9.15 ns	**1085.29**	8.01ns
Decoder	N/A	N/A	344.20	2.73ns
Total	4868.60	9.15ns (worst)	3612.14	**8.01ns** (worst)

6. SUMMARY AND CONCLUSIONS

We presented several control schemes for maximizing the compression effectiveness and reducing the test data in iBIST environment. Benchmark results show that the control schemes using selective polarity inversion and static control is superior to all the presented control schemes. In addition, we described the overall iBIST controller implementation including decoder logic. The results indicate that iBIST uses lower overhead area and produces higher performance than traditional BIST.

iBIST is a viable solution for overcoming the test data bandwidth gap between external I/Os and the internal circuit. It also promotes the design and test reuse and provides the protection of IP cores via control vectors. Its main benefit is achieving a high test quality within optimal test execution time without inserting test points. The decompression technique is independent of the design and promotes reusability at low test overhead.

8. REFERENCES

[1] A. Chandra and K. Chakrabarty, “Test data compression for system-on-a-chip using Golomb codes,” *Proc. IEEE VLSI Test Symposium*, pp. 2000.

[2] A. Chandra and K. Chakrabarty, “On Using Golomb Codes and Internal Scan Chains for Test Data Compression / Decompression in a System-on-a-Chip,” *Proc. of Testing Embedded Core-based System-Chips Workshop*, pp. 2.2-1 ~ 2.2-7, 2000.

[3] Joseph Desposito. "SOC and Deep-Submicron Technology Drive New DFT Strategies," *Electronic Design*, August 3, 1998. pp. 49-62.

[4] Rajesh K. Gupta and Yervant Zorian, "Introducing Core-Based System Design," *IEEE Design and Test of Computers*, October-December, 1997.

[5] Hellebrand S., et al., "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters," IEEE Proc. of Int’l Test Conference, pp 778-984, Oct. 2000

[6] M. Ishida, D.D. Ha, T. Yamaguchi, “COMPACT: A Hybrid Method for Compressing Test Data,” Proc. Of VLSI Test Symposium, pp.62-69, 1998

[7] A. Jas, J. Ghosh_Dastidar, and N.A. Touba, “Scan vector compression/decompression using statistical coding,” Proc. IEEE VLSI Test Symposium, pp. 114-120, 1999

[8] D. Kay and S. Mourad, “Controllable LFSR for Built-In Self-Test” Proc. of IEEE Instrument and Measurement Technology Conference, pp 223-229, May 1-4, 2000

[9] D. Kay and S. Mourad, “Controllable LFSR for Embedded Core BIST”, Proc. Of Testing Embedded Core-based System-Chips Workshop, pp. 2.4-1 ~ 2.4-6, 2000

[10] D. Kay and S. Mourad, “Compression Technique for Interactive BIST Application”, Proc. Of VLSI Test Symposium, pp. 9-14 2001

[11] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," Proc. of European Test Conference, pp. 237-242, 1991

[12] Muradali, F., V.K. Agrawal, B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test," proc. of International Test Conference, pp. 660-668, 1990.

[13] Bahram Pouya and Nur A. Touba. "Modifying user-defined logic for test access to embedded cores." International Test Conference, 1997 IEEE pp.60-68.

[14] SCU-RTL Benchmarks. http://www.engr.scu.edu/mourad

[15] Nur A. Touba and Edward J. McCluskey, “Altering a pseudo random bit sequence for scan based bist” International Test Conference, 1996 IEEE pp.167-175

[16] Venkataraman, S., et al. "An Efficient BIST Scheme based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," Proc. of International Conference on Computer-Aided Design (ICCAD), pp 572-577, 1993

[17] H.-J. Wunderlich, G. Kiefer: "Bit-Flipping BIST", Proc. Int. Conf. On CAD, pp 337-343, 1996

[18] T. Yamaguchi, M. Tilgner, M. Ishida, and D.S. Ha, “An Efficient Method for Compressing Test Data,” Proceedings of International Test Conference, pp.79-88, November 1997