IEEE International Test Conference, Washington, DC, October 1996

# MIXED-MODE BIST USING EMBEDDED PROCESSORS

Sybille Hellebrand, Hans-Joachim Wunderlich, Andre Hertwig
Institute of Computer Structures
University of Siegen, Germany

## Abstract

*In complex systems, embedded processors may be used to run software routines for test pattern generation and response evaluation. For system components which are not completely random pattern testable, the test programs have to generate deterministic patterns after random testing. Usually the random test part of the program requires long run times whereas the part for deterministic testing has high memory requirements.*

*In this paper it is shown that an appropriate selection of the random pattern test method can significantly reduce the memory requirements of the deterministic part. A new, highly efficient scheme for software-based random pattern testing is proposed, and it is shown how to extend the scheme for deterministic test pattern generation. The entire test scheme may also be used for implementing a scan based BIST in hardware.*

## 1 Introduction

Integrating complex systems into single chips or implementing them as multi-chip modules (MCMs) has become a widespread approach. A variety of embedded processors and other embedded coreware can be found on the market, which allows to appropriately split the system functionality into both hardware and software modules. With this development, however, system testing has become an enormous challenge: the complexity and the restricted accessibility of hardware components require sophisticated test strategies. Built-in self-test combined with the IEEE 1149 standards can help to tackle the problem at low costs [10].

For conventional ASIC testing, a number of powerful BIST techniques have been developed in the past [1 - 3, 5, 7 - 8, 19, 26, 29 - 31]. For example, it has been shown that combining random and efficiently encoded deterministic patterns can provide complete fault coverage while keeping the costs for extra BIST hardware and the storage requirements low [13, 14, 32]. In the case of embedded systems such a high quality test is possible without any extra hardware by just using the embedded processor to generate the tests for all other components.

Usually, this kind of functional testing requires large test programs, and a memory space not always available on the system. In this paper it will be shown how small test programs can be synthesized such that a complete coverage of all non-redundant stuck-at faults in the combinational parts of the system is obtained. The costs for extra BIST hardware in conventional systems testing are reduced to the costs for some hundred bytes of system memory to store the test routines. The proposed BIST approach can efficiently exploit design-for-testability structures of the subcomponents. As shown in Figure 1 during serial BIST the embedded processor executes a program which generates test patterns and shifts them into the scan register(s) of the component(s) to be tested. Even more efficiently, the presented approach may be used to generate test data for input registers of pipelined or combinational subsystems.

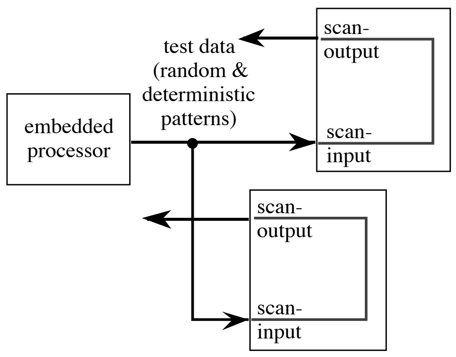


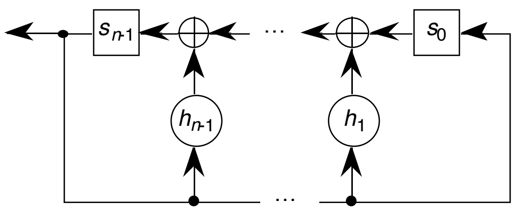
Figure 1: Serial BIST approach.

The structure of the test program can be kept very simple, if only random patterns have to be generated, since then some elementary processor instructions can be used [12, 21, 25, 28].

Even a linear feedback shift register (LFSR) can be emulated very efficiently: Figure 2 shows as an example a

This work has been supported by the DFG grant „Test und Synthese schneller eingebetteter Systeme“ (Wu 245/1-2).

modular LFSR and the corresponding program (for simplicity the C-code is shown) to generate a fixed number of state transitions.



```
void transition (int m, int n,
                 unsigned int polynomial,
                 unsigned int *state)
/* m transitions of modular LFSR of degree n */
{
  int i;
  for (i=0; i<m; i++)
  {
    if (*state >> n-1)
    {
      *state <<= 1;
      *state ^= polynomial;
    }
    else
      *state <<= 1;
  }
}
```

Figure 2: Modular LFSR and corresponding program for generating state transitions.

But usually not all the subcomponents of a system will be random pattern testable, and for the remaining faults deterministic test patterns have to be applied. For this purpose, compact test sets may be generated as described in [16, 18, 22, 27] and reproduced by the test program, or a hardware-based deterministic BIST scheme is emulated by the test software [13 - 15, 32]. This kind of mixed-mode testing may interleave deterministic and random testing or perform it successively. In each case, the storage requirements for the deterministic part of the test program are directly related to the number of undetected faults after random pattern generation. There is a great trade-off between the run-time for random test and the memory requirements of the mixed-mode program. Assume a small improvement of the random test method which leads to an increase of the fault coverage from 99.2% to 99.6%. This reduces the number of undetected faults and the storage requirements by the factor 1/2. Overall, the efficiency of a mixed-mode test scheme can be improved to a much higher degree by modifying its random part rather than its deterministic part.

In this paper a highly efficient software-based random BIST scheme is presented which is also used for generating deterministic patterns. The rest of the paper is organized as follows: In the next section, different random pattern test schemes to be emulated by software are evaluated, and in section 3 the extension to deterministic testing is described. Subsequently, in section 4, a procedure for optimizing the overall BIST scheme is presented, and section 5 describes the procedure for generating the mixed-mode test program. Finally, section 6 gives some experimental results based on the INTEL 80960CA processor as an example.

## 2 Emulated Random Pattern Test

Test routines exploiting the arithmetic functions of a processor can produce patterns with properties which are sufficient for testing random pattern testable circuits [12, 25], even if they do not completely satisfy all the conditions for randomness as stated in [11], e.g.. However, for other circuits, in particular for circuits considered as random pattern resistant, arithmetic patterns may not perform as well. Linear feedback shift registers (LFSRs) corresponding to primitive feedback polynomials and cellular automata are generally considered as stimuli generators with good properties for random testing [9, 17, 20]. But the generated sequences still show some linear dependencies, such that different primitive polynomials perform differently on the same circuit. In some cases, the linear dependencies may support fault detection, for other circuits they perform poorly. In the following, the fault coverage obtained by several LFSR-based pattern generation schemes will be discussed with some experimental data.

### *2.1 Feedback Polynomial*

In contrast to hardware-based BIST, in a software-based approach the number and the positions of the feedback taps of the LFSR have no impact on the costs of the BIST implementation. Thus, for a given length the achievable fault coverage can be optimized without cost constraints.

Assuming a test per scan scheme as shown in Figure 3 the sensitivity of the fault coverage to the selected feedback polynomial has been studied by a series of experiments for the combinational parts of the ISCAS85 and ISCAS89 benchmark circuits [4, 6].

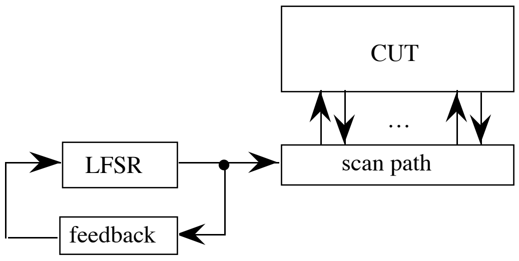


Figure 3: Scan-based BIST.

| Circuit | PI | F | Degree | LFSR1 | LFSR2 | LFSR3 | LFSR4 | LFSR5 | LFSR6 | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| c2670 | 157 | 2478 | 52 | 12.55<br>99.68 | 11.99<br>95.23 | **12.59**<br>**100** | **11.74**<br>**93.25** | 12.23<br>97.14 | 12.47<br>99.05 | 12.26<br>97.39 |
| c3540 | 50 | 3291 | 17 | **0.03**<br>**20** | 0.12<br>80 | 0.06<br>40 | **0.15**<br>**100** | 0.09<br>60 | 0.09<br>60 | 0.09<br>60 |
| s420.1 | 34 | 455 | 20 | **18.90**<br>**100** | 12.97<br>68.62 | **8.79**<br>**46.51** | 16.70<br>88.36 | 15.82<br>83.70 | 10.11<br>53.49 | 13.88<br>73.45 |
| s641 | 54 | 465 | 22 | 2.58<br>46.15 | 1.72<br>30.77 | **5.59**<br>**100** | 2.15<br>38.46 | **1.51**<br>**27.01** | 1.94<br>34.70 | 2.58<br>46.18 |
| s838.1 | 66 | 931 | 37 | 33.73<br>89.21 | 37.49<br>99.15 | 34.91<br>92.33 | **37.81**<br>**100** | **33.08**<br>**87.49** | 37.59<br>99.42 | 35.77<br>94.6 |
| s9234 | 247 | 6475 | 52 | 9.87<br>89.16 | 10.83<br>97.83 | 10.75<br>97.11 | **9.37**<br>**84.64** | **11.07**<br>**100** | 9.93<br>89.70 | 10.30<br>93.07 |

Table 1: Absolute and normalized (w. r. t. worst LFSR) percentage of undetected non-redundant faults after 10,000 patterns.

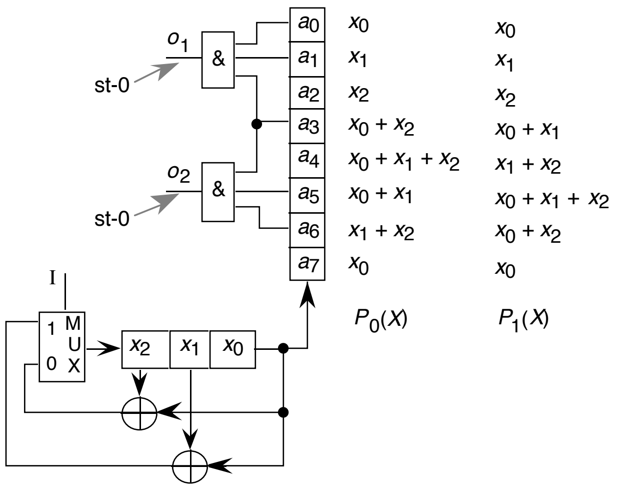
Fault simulation of 10,000 random patterns was performed for each circuit using several different feedback polynomials, all of the same degree. Some typical results are shown in Table 1. The first four columns contain the circuit name, the number of inputs, the number of non-redundant faults, and the selected degree of the feedback polynomial.[1] The remaining columns show the characteristics for six different LFSRs. The first entry reports the percentage of undetected non-redundant faults, and the second entry normalizes this number to the corresponding number for the worst LFSR (in %). The worst and best performing LFSR are printed in bold, respectively. The last column gives the average over all of the LFSRs.

It can be observed that there is a big variance in the performance of different LFSRs of the same degree. For s641, e.g., the best LFSR reduces the number of undetected faults down to 27% of the faults left undetected by the worst polynomial.

## 2.2 *Multiple-Polynomial LFSRs*

One explanation for the considerable differences in fault coverage observed in section 2.1 is given by the fact, that linear dependencies of scan positions may prevent certain necessary bit combinations in the scan patterns independent of the initial state of the LFSR [2]. For different LFSRs the distribution of linear dependencies in the scan chain is different and, depending on the structure of the circuit, may have a different impact on the fault coverage.

As shown in Figure 4 the impact of linear dependencies can be reduced if several polynomials are used. In this small example the LFSR can operate according to two different primitive feedback polynomials $P_0(X) = X^3 + X^2 + 1$ and $P_1(X) = X^3 + X + 1$, which are selected by the input of the multiplexer. For any given initial state $(x_0, x_1, x_2)$ the LFSR produces a scan pattern $(a_0, \ldots, a_7)$, such that, depending on the selected polynomial, the shown equations for $P_0(X)$ or $P_1(X)$ hold for its components.



$o_1$
&
st-0
$o_2$
&
st-0
I
1 MUX 0
$x_2$ $x_1$ $x_0$
$a_0$ $a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$
$x_0$, $x_1$, $x_2$, $x_0 + x_2$, $x_0 + x_1 + x_2$, $x_0 + x_1$, $x_1 + x_2$, $x_0$
$x_0$, $x_1$, $x_2$, $x_0 + x_1$, $x_1 + x_2$, $x_0 + x_1 + x_2$, $x_0 + x_2$, $x_0$
$P_0(X)$
$P_1(X)$

Figure 4: Scan-based BIST with multiple-polynomial LFSR.

For polynomial $P_0$ there is a linear relation $a_3 + a_5 = x_0 + x_2 + x_0 + x_1 = x_1 + x_2 = a_7$, which prevents the combination (1, 1, 1) at the inputs of the AND-gate. This implies that the polynomial $P_0(X)$ can never produce a test for the stuck-at-0 fault at node $o_2$. In contrast to that, for polynomial $P_1(X)$ the same input positions are linearly independent and produce all possible nonzero bit combinations and thus a test for the considered fault. Similarly, the stuck-at-0 fault at node $o_1$ cannot be tested using polynomial $P_1(X)$, but polynomial $P_0(X)$ can provide a test. Using both polynomials, each for a certain number of patterns, increases the chance of detecting both faults.

1 The degrees of the polynomials have been selected, such that they were compatible with the requirements for the deterministic test described in section 3.

Such a multiple-polynomial LFSR can be implemented efficiently in hardware by trying to share parts of the feedback for several polynomials. A software emulation is also very simple, since the basic procedure to simulate an LFSR has to be modified only slightly. To control the selection of feedback polynomials several schemes are possible. The first is shown in Figure 5 assuming *N* random patterns to be generated by p different polynomials $P_i$, $i = 0, ..., p-1$. LFSR($P_i$) denotes the LFSR operation corresponding to feedback polynomial $P_i$.

```
initialize (LFSR);
for (i = 0; i < p; i++)
  generate ⌈N/p⌉ patterns by LFSR(P_i);
```

Figure 5: Successive multiple-polynomial scheme (SUC).

The polynomials are applied successively to generate contiguous subsequences of $\lceil N/p \rceil$ random patterns, the scheme will therefore be referred to as scheme SUC. For one polynomial the scheme degenerates to the conventional single polynomial scheme. The possibility to switch between different distributions of linear dependencies is paid by the disadvantage that some patterns may occur repeatedly up to *p* times. Hence, an overall increase of the fault coverage cannot be expected, but experiments have shown that there is indeed an improvement for some circuits. Table 2 lists the results for the same set of circuits as studied in the previous section.

| Circuit | Degree | p = 2 | p = 3 | p = 4 | p = 5 |
|---|---|---|---|---|---|
| c2670 | 52 | 12.55<br>99.68<br>106.9 | 12.55<br>99.68<br>106.9 | 8.76<br>**69.58**<br>**74.62** | 12.55<br>99.68<br>106.9 |
| c3540 | 17 | 0.03<br>**20**<br>**100** | 0.12<br>80<br>400 | 0.09<br>60<br>300 | 0.12<br>80<br>400 |
| s420.1 | 20 | 14.95<br>79.10<br>170.08 | 14.51<br>76.77<br>165.07 | 13.41<br>70.95<br>152.56 | 12.97<br>**68.62**<br>**147.55** |
| s641 | 22 | 1.94<br>34.70<br>128.48 | 1.94<br>34.70<br>128.48 | 1.94<br>34.7<br>128.48 | 1.72<br>**30.77**<br>**113.91** |
| s838.1 | 37 | 30.72<br>81.25<br>92.87 | 31.26<br>82.68<br>94.5 | 31.26<br>82.68<br>94.5 | 30.18<br>**79.82**<br>**91.23** |
| s9234 | 52 | 10.61<br>95.84<br>113.23 | 10.67<br>96.39<br>113.87 | 10.13<br>91.51<br>108.11 | 9.22<br>**83.29**<br>**98.40** |

Table 2: Absolute and normalized (w. r. t. worst and best single LFSR) percentage of undetected non-redundant faults for scheme SUC after 10,000 patterns.

For each circuit 10,000 patterns were simulated using $p = 2, \ldots, 5$ polynomials. For each experiment the percentage of undetected non-redundant faults is reported (1st line), as well as the corresponding normalized numbers for the worst (2nd line) and for the best single polynomial (3rd line) of the same degree (in %).

Applying the successive scheme for example to the circuit c2670 with $p = 4$ reduces the number of undetected faults down to 69.58% compared with the worst single polynomial. Even more important is that the scheme also outperforms the best single polynomial and the number of remaining target faults for ATPG is less than 75%, i.e. 25% percent of the faults left by the best single polynomial are additionally covered by this scheme.

The randomness of the sequence can be further increased, if the polynomials are not used successively, but selected randomly for each test pattern. This random selection can be implemented by a second LFSR as shown in Figure 6 and will be referred to as scheme RND.

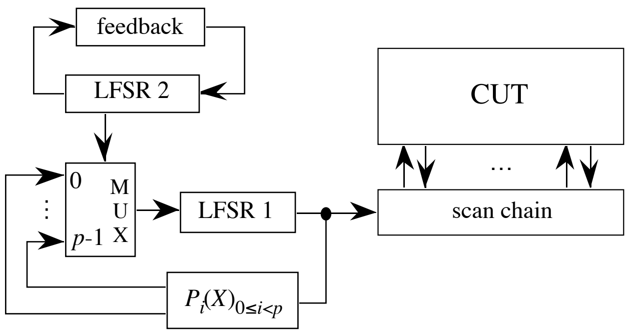


Figure 6: Hardware scheme for the random selection of feedback polynomials (RND).

The selection between *p* different feedback polynomials for LFSR1 is controlled by $\lceil \log_2 p \rceil$ bits of the state register of LFSR2. For a software implementation of the structure of Figure 6, two additional registers are required for storing the feedback polynomial and the state of LFSR2. LFSR1 and LFSR2 can be emulated by the same procedure, and the complete routine to generate a sequence of *N* random patterns is shown in Figure 7.

```
initialize (LFSR1);
initialize (LFSR2);
for (i = 0; i < N; i++)
{
  select P based on state of LFSR2;
  generate 1 pattern by LFSR1(P);
  perform 1 state transition of LFSR2;
}
```

Figure 7: Software routine for the random pattern generation scheme of Figure 6 (RND).

Table 3 shows the percentage of undetected non-redundant faults and the corresponding normalized numbers obtained by the scheme RND for $p = 2, ..., 5$ feedback polynomials.

| Circuit | Degree | p = 2 | p = 3 | p = 4 | p = 5 |
|---|---|---|---|---|---|
| c2670 | 52 | 12.63<br>100.32<br>107.58 | 12.35<br>98.09<br>105.2 | 11.99<br>**95.23**<br>**102.13** | 12.55<br>99.68<br>106.9 |
| c3540 | 17 | 0.06<br>**40**<br>**200** | 0.09<br>60<br>300 | 0.09<br>60<br>300 | 0.12<br>80<br>400 |
| s420.1 | 20 | 12.75<br>**67.46**<br>**145.05** | 14.51<br>76.77<br>165.07 | 14.29<br>75.61<br>162.57 | 17.14<br>90.69<br>194.99 |
| s641 | 22 | 1.72<br>30.77<br>113.91 | 1.94<br>34.7<br>128.48 | 1.94<br>34.7<br>128.48 | 1.51<br>**27.01**<br>**100** |
| s838.1 | 37 | 38.56<br>101.98<br>116.57 | 33.40<br>**88.34**<br>**100.97** | 36.95<br>97.73<br>111.7 | 36.84<br>97.43<br>111.37 |
| s9234 | 52 | 9.61<br>**86.81**<br>**102.56** | 11.24<br>101.54<br>119.96 | 10.16<br>91.78<br>108.43 | 10.75<br>97.11<br>114.73 |

Table 3: Absolute and normalized (w. r. t. worst and best single LFSR) percentage of undetected non-redundant faults for scheme RND after 10,000 patterns.

For the randomly selected polynomials, there is a higher chance of pattern repetitions, but randomly switching between different distributions of linear dependencies may improve the quality of the patterns. For some circuits, this results in an improvement of fault coverage, so that the set of faults which remain for deterministic testing is further reduced.

### 2.3 Multiple-Polynomial, Multiple-Seed LFSRs

Another way of improving the efficiency of a random test is repeatedly storing a new seed during pattern generation as investigated for instance in [23]. This technique can be combined with the use of multiple polynomials as shown in Figure 8.

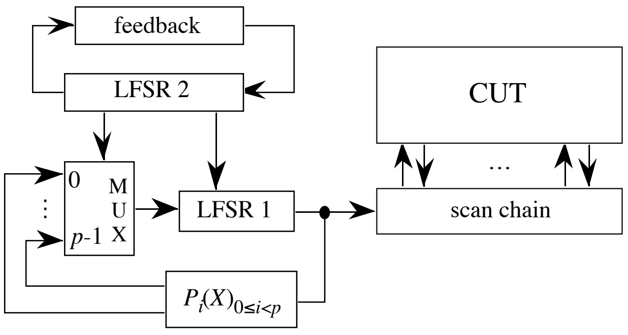


Figure 8: Multiple-polynomial, multiple-seed LFSR.

As for the scheme RND, $\lceil \log_2 p \rceil$ bits of the state register of LFSR2 are used to drive the selection between $p$ different feedback polynomials of degree $k$ for LFSR1. The remaining $k$ bits provide the seed for LFSR1. In the sequel this scheme will be referred to as the scheme RND$^2$. The structure of the corresponding test program is shown in Figure 9.

```
initialize (LFSR2);
for (i = 0; i < N; i++)
{
  select seed S and polynomial P
    based on state of LFSR2;
  initialize LFSR1 with S;
  generate 1 pattern by LFSR1(P);
  perform 1 state transition of LFSR2;
}
```

Figure 9: Test program for the multiple-polynomial, multiple-seed LFSR (RND$^2$).

Again, in this scheme patterns may occur repeatedly, but in addition to the advantage of randomly changing the distribution of linear dependencies this scheme is also able to generate the all zero-vector which is often needed for complete fault coverage.

Table 4 gives the results for $p$ = 2, ..., 5 polynomials (percentage of undetected non-redundant faults and the corresponding normalized numbers as in Tables 2 and 3).

| Circuit | Degree | p = 2 | p = 3 | p = 4 | p = 5 |
|---|---|---|---|---|---|
| c2670 | 52 | 12.31<br>97.78<br>104.86 | 12.15<br>**96.51**<br>**103.49** | 12.15<br>96.51<br>103.49 | 12.55<br>99.68<br>106.9 |
| c3540 | 17 | 0.12<br>**80**<br>**400** | 0.18<br>120<br>600 | 0.18<br>120<br>600 | 0.18<br>120<br>600 |
| s420.1 | 20 | 12.31<br>65.13<br>140.05 | 13.19<br>69.79<br>150.06 | 12.75<br>67.46<br>145.05 | 10.99<br>**58.15**<br>**125.03** |
| s641 | 22 | 1.94<br>**34.7**<br>**128.48** | 1.94<br>34.7<br>128.48 | 1.94<br>34.7<br>128.48 | 2.15<br>38.46<br>142.38 |
| s838.1 | 37 | 27.71<br>73.29<br>83.77 | 23.52<br>**62.21**<br>**71.1** | 23.52<br>62.21<br>71.1 | 26.53<br>70.17<br>80.2 |
| s9234 | 52 | 9.14<br>**82.57**<br>**97.55** | 9.85<br>88.98<br>105.12 | 9.58<br>86.54<br>102.24 | 9.58<br>86.54<br>102.24 |

Table 4: Absolute and normalized (w. r. t. worst and best single LFSR) percentage of undetected non-redundant faults for scheme RND$^2$ after 10,000 patterns.

As expected, not for all circuits the fault coverage increases, but there are circuits where this technique leads to significant improvements. For circuits s838.1 and s9234 the best results are obtained compared with all the experiments before.

## 3 Software-Based Deterministic BIST

The structure of the multiple-polynomial, multiple-seed random BIST scheme of Figure 8 is very similar to the deterministic BIST scheme based on reseeding of multiple-polynomial LFSRs proposed in [13, 14], see Figure 10.

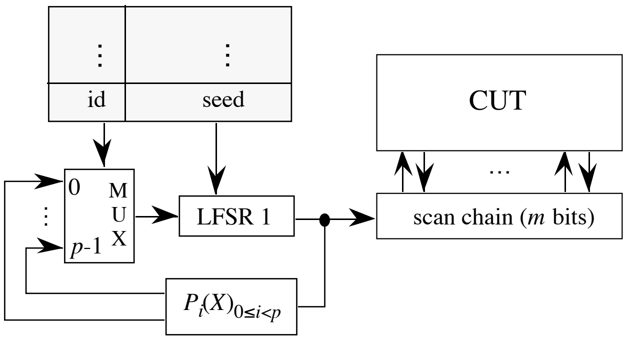


Figure 10: Deterministic BIST scheme based on a multiple-polynomial LFSR by [14].

A deterministic pattern is encoded as a polynomial identifier and a seed for the respective polynomial. During test mode the pattern can be reproduced by emulating the LFSR corresponding to the polynomial identifier, loading the seed into the LFSR and performing $m$ autonomous transitions of the LFSR. After the $m$-th transition the scan chain contains the desired pattern which is then applied to the CUT.

To calculate the encoding systems of linear equations have to be solved. For a fixed feedback polynomial $h(X) = X^k + h_{k-1}X^{k-1} + \dots + h_1X + h_0$ of degree $k$ the LFSR produces an output sequence $(a_i)_{i\geq 0}$ satisfying the feedback equation $a_i = a_{i-1}\cdot h_{k-1} + \dots + a_{i-k}\cdot h_0$ for all $i \geq k$. The LFSR-sequence is compatible with a desired test pattern $t = (t_1, ..., t_m)$, if for all specified bits $a_i = t_i$ holds. Recursively applying the feedback equation provides a system of linear equations in the seed variables $a_0, ..., a_{k-1}$. If no solution can be found for the given polynomial, the next available polynomial is tried, and in [14] it has been shown that already for 16 polynomials there is a very high probability of success that a deterministic pattern with $s$ specified bits can be encoded into an $s$-bit seed.

Hence, if $p$ different polynomials are available and the polynomial identifier is implemented as a „next bit“, the seed and the next bits for a deterministic test set $T = \{t_1,...,t_N\}$ with maximum number of specified bits $s_{max}$ require $S(T) := (s_{max} + 1)\cdot N$ bits of storage. Minimizing $S(T)$ requires both minimizing the maximum number of care bits $s_{max}$ and the number of patterns $N$. In [15] an ATPG-algorithm was presented which generates test patterns where the number of specified bits $s_{max}$ is minimized. In a mixed-mode BIST approach the number $N$ of patterns is highly correlated to the number of faults left undetected after random testing.

## 4 Synthesizing the BIST Scheme

Since the efficiency of a mixed-mode BIST scheme strongly depends on the number of hard faults to be covered by deterministic patterns, a major concern in synthesizing the BIST scheme is optimizing the random test. The experimental data of section 2 show that significant variances in the fault efficiency achieved by different LFSR schemes exist, and that there is no universal scheme or polynomial working for all of the circuits. In the sequel, a procedure is presented for determining an optimized LFSR scheme. The selection of the LFSRs is guided, such that the fault efficiency is maximized while satisfying the requirements for an efficient encoding of deterministic patterns for the random pattern resistant faults. Assuming a table of primitive polynomials available the proposed procedure consists of 4 steps:

1) Perform ATPG to eliminate the redundant faults and to estimate the maximum number of specified bits, $s_{max}$, to be expected in the test cubes for the hard faults.
2) Select $M$ polynomials of degree $s_{max}$ randomly, and perform fault simulation with the corresponding shift register sequences. Rank the polynomials according to the fault coverage achieved.
3) Select the $P$ best polynomials and store the highest fault coverage and the corresponding LFSR as BEST_SCHEME.
4) Using $2 \leq p \leq P$ polynomials, simulate the schemes SUC, RND, and RND$^2$. Update BEST_SCHEME to the best solution obtained so far.

The number $M$ is mainly determined by a limit of the computing time to be spent. The number $P$ is also restricted by the computing time available, but in addition to that each LFSR requires two registers of the processor for pattern generation. So, the register file of the target processor puts a limit on $P$, too.

Table 5 shows the results achieved by this procedure for the same set of circuits as studied in section 2. For the same degrees as used in section 2 sequences of 10,000 random patterns were applied.

| Circuit | Best Scheme | $p$ | FE | UF | UF$_{best}$ | UF$_{worst}$ |
|---|---|---|---|---|---|---|
| c2670 | SUC | 4 | 91.24 | 8.76 | 74.62 | 69.58 |
| c3540 | SUC | 2 | 99.97 | 0.03 | 100 | 20 |
| s420.1 | SUC | 1 | 91.21 | 8.79 | 100 | 63.33 |
| s641 | RND | 5 | 98.49 | 1.51 | 100 | 27.01 |
| s838.1 | RND$^2$ | 3 | 76.48 | 23.52 | 71.10 | 62.21 |
| s9234 | RND$^2$ | 2 | 90.86 | 9.14 | 97.55 | 82.57 |

Table 5: Best schemes and relation to best and worst single polynomial solution.

The second and third column show the best scheme and the corresponding number of polynomials $p$, column 4 provides the fault efficiency FE (percentage of detected non-redundant faults). The percentage of faults left undetected by the best scheme is reported in column UF. $UF_{best}$ normalizes this solution to the number obtained by the best single polynomial, $UF_{worst}$ refers to the worst single polynomial.

Table 5 indicates that the search for an appropriate random test scheme can reduce the number of remaining faults significantly. The procedure needs $M + 3 \cdot (P - 1)$ runs of fault simulations, but may decrease the storage amount needed for deterministic patterns considerably. These savings in memory for the mixed-mode test program are particularly important, if the test program has to be stored in a ROM for start-up and maintenance test.

## 5 Generating Mixed-Mode Test Programs

Test programs implementing the random test schemes and the reseeding scheme for deterministic patterns were generated for the INTEL 80960CA as a target processor. Its large register set made a very compact coding possible. Since the part of the test program which generates the deterministic patterns is a superset of instructions required for implementing any of the random schemes, only the example for the most complex random scheme is shown. The mixed-mode test program of Figure 11 generates random test patterns by multiple-polynomial, multiple-seed LFSR emulation, and switches to the reseeding scheme afterwards.

The program of Figure 11 requires 27 words in memory but assumes that all LFSRs fit into 32 bit registers. This

```
steps1        equ      ...                           ; number of steps for lfsr1
steps2        equ      ...                           ; number of steps for lfsr2
steps_det     equ      ...                           ; number of steps for deterministic test
len1          equ      ...                           ; position of msb of lfsr1
len2          equ      ...                           ; position of msb of lfsr2
testport      equ      ...                           ; address of testport
no_poly_bits  equ      ...                           ; number of bits for polynomial choice
mask          equ      ...                           , define mask
start         dq       startvector                   ; define startvector for lfsr2
poly          dq       polynomials                   ; define polynomials for lfsr1
                                                     ; and lfsr2 (poly[0])
seeds         dq       seedvectors                   ; define seeds for det. test
seed_offset   equ      seeds - start                 ; define offset for seed table

begin:        lda      testport, r10                 ; load address of testport
              lda      steps_det, r11                ; load loopcounter for lfsr1 in det. mode
              lda      steps1, r12                   ; load loop counter for lfsr1
              lda      start, r14                    ; load startvector address for lfsr1
              ld       (r14), r6                     ; load startvector for lfsr2
              ld       4(r14), r7                    ; load polynomial for lfsr2
l0:           mov      r6, r4                        ; initialize lfsr1 with contents of lfsr2
              and      mask, r4, r15                 ; compute poly-id
              ld       8(r14)[r15*4], r5             ; polynomial for lfsr1
              lda      no_poly_bits, r15             ; load number of bits for poly-id
l1:           shro     no_poly_bits, r4, r4          ; shift poly-bits
              lda      steps2, r13                   ; load loop counter for lfsr1
l2:           st       r4, (r10)                     ; write testpattern to testport
              mov      r4, r8
              shlo     1, r8, r4                     ; shift left
              bbc      len2, r8, l3                  ; branch if msb of lfsr2 equal zero
              xor      r4, r5, r4                    ; xor
l3:           subi     r13, 1, r13                   ; decrement loop counter
              cmpibne  r13, 0, l2                    ; branch not equal zero
              mov      r6, r8
              shlo     1, r8, r6                     ; shift left
              bbc      len1, r8, l4                  ; branch if msb of lfsr1 equal zero
              xor      r6, r7, r6                    ; xor
l4:           subi     r12, 1, r12                   ; decrement loop counter
              cmpibg   r12, r11, l0                  ; branch if r12 > steps_det
              ld       seed_offset(r14)[r12*4],r6    ; load seed
              cmpibne  r12,0,l0
```

Figure 11: Mixed-mode BIST program.

is always possible for random pattern generation, but encoding deterministic patterns may lead to LFSR lengths exceeding 32 bits. In this case, the program of Figure 11 has to be modified in a straightforward way, and requires more memory. Table 6 gives the relation between memory requirements and LFSR lengths.

| LFSR length | 32 | 64 | 96 | 128 |
|---|---|---|---|---|
| Memory requirements (words) | 27 | 41 | 52 | 63 |

Table 6: LFSR length and memory requirements for the mixed-mode test program.

In addition to the program size, memory has to be reserved for storing the polynomials and the seeds in order to decode the deterministic patterns. The experimental results of the next section show that these data form by far the major part of the memory requirements.

## 6 Experimental Results

The described strategy for generating mixed-mode test programs was applied to all the benchmark circuits for $M = 16$ and $P = 5$, i. e. for each circuit $M + 3\cdot(P - 1) = 28$ runs of fault simulation were performed to determine the best random scheme. Tables 7 and 8 show the results.

| Circuit | PI | Degree | Best Scheme | *p* |
|---|---|---|---|---|
| c2670 | 157 | 52 | SUC | 4 |
| c3540 | 50 | 19 | SUC | 2 |
| c7552 | 206 | 106 | RND$^2$ | 3 |
| s420.1 | 34 | 20 | SUC | 1 |
| s641 | 54 | 22 | SUC | 1 |
| s713 | 54 | 22 | SUC | 1 |
| s820 | 23 | 15 | SUC | 5 |
| s832 | 23 | 15 | RND$^2$ | 5 |
| s838.1 | 66 | 37 | RND$^2$ | 3 |
| s953 | 45 | 15 | SUC | 1 |
| s1196 | 32 | 17 | RND$^2$ | 2 |
| s1238 | 32 | 17 | RND$^2$ | 2 |
| s1423 | 91 | 25 | RND$^2$ | 5 |
| s5378 | 214 | 25 | RND | 5 |
| s9234 | 247 | 52 | RND$^2$ | 2 |
| s13207 | 700 | 60 | SUC | 5 |
| s15850 | 611 | 48 | SUC | 2 |
| s38417 | 1664 | 106 | RND$^2$ | 4 |
| s38584 | 1464 | 60 | SUC | 2 |

Table 7: Circuit characteristics and best random scheme.

The selected random schemes and their characteristic data are reported in Table 7. Columns 2 and 3 list the number of primary inputs PI and the degree of the polynomials. The best random scheme and the number of polynomials *p* are reported in the subsequent columns.

Table 8 shows the detailed results. The number of nonredundant faults for each circuit is given in column 2. The efficiency of the random scheme is characterized again by the fault efficiency FE, the percentage of undetected nonredundant faults UF and the normalized numbers for UF with respect to the best ($UF_{best}$) and the average ($UF_{average}$) single polynomial solution in columns 3 through 6.

| Circuit | F | FE | UF | $UF_{best}$ | $UF_{average}$ |
|---|---|---|---|---|---|
| c2670 | 2478 | 91.24 | 8.76 | 74.62 | 71.45 |
| c3540 | 3291 | 99.97 | 0.03 | 100 | 33.33 |
| c7552 | 7419 | 98.87 | 1.13 | 30.46 | 26.84 |
| s420.1 | 455 | 91.21 | 8.79 | 100 | 63.33 |
| s641 | 465 | 98.49 | 1.51 | 100 | 27.01 |
| s713 | 543 | 98.71 | 1.29 | 100 | 70.11 |
| s820 | 850 | 100 | - | - | - |
| s832 | 856 | 99.77 | 0.23 | 24.73 | 4.21 |
| s838.1 | 931 | 76.48 | 23.52 | 71.1 | 65.75 |
| s953 | 1079 | 99.26 | 0.74 | 100 | 18.5 |
| s1196 | 1242 | 99.28 | 0.72 | 68.57 | 40.91 |
| s1238 | 1286 | 99.38 | 0.62 | 66.67 | 35.43 |
| s1423 | 1501 | 100 | - | - | - |
| s5378 | 4563 | 99.45 | 0.55 | 85.94 | 61.8 |
| s9234 | 6475 | 90.86 | 9.14 | 97.55 | 88.74 |
| s13207 | 9664 | 94.45 | 5.55 | 89.95 | 80.79 |
| s15850 | 11336 | 94.89 | 5.11 | 94.28 | 89.96 |
| s38417 | 31015 | 93.92 | 6.08 | 92.26 | 85.75 |
| s38584 | 34797 | 98.77 | 1.23 | 95.35 | 82.55 |

Table 8: Fault efficiency and percentage of undetected nonredundant faults for the best random schemes after 10,000 patterns.

The reduction of the remaining faults obtained by the best random test scheme is significant. For instance, the circuit c7552 is known to be very random pattern resistant, and a single polynomial solution in the average leads to a fault efficiency of 95.79% leaving 4.21% of the faults for deterministic encoding. For the same circuit, the RND$^2$ scheme achieves a fault efficiency of 98.87%, and only 1.13% or, absolutely, 84 faults are left. This corresponds to a reduction of the remaining faults down to 27%.

For circuits s820 and s1423 a careful selection of the random scheme even makes the deterministic test superfluous. Finally, it should be noted that for the larger cir cuits already a small relative reduction means a considerable number of faults which are additionally covered by the random test and need not be considered during the deterministic test. For example for circuit s38417 a reduction down to 85.75% and 92.26%, respectively, means that additional 313 and 158, respectively, faults are eliminated during random test.

Table 9 shows the resulting number of test patterns required for the random pattern resistant faults and the amount of test date storage (in bits) for the best random scheme compared to a random test using an average single polynomial. This includes the storage needed for the polynomials, the initial LFSR states for the random test and the encoded deterministic test set. Since the goal of this work was to determine the impact of the random test on the test data storage, a standard ATPG tool was selected to perform the experiments [24]. For all circuits the fault efficiency is 100% after the deterministic test.

| Circuit | Deterministic patterns | | Test data storage (bits) | |
|---|---|---|---|---|
| | Best scheme | Average polynomial | Best scheme | Average polynomial |
| c2670 | 73 | 77 | 4186 | 4239 |
| c3540 | 1 | 1 | 59 | 59 |
| c7552 | 51 | 92 | 6889 | 11644 |
| s420.1 | 22 | 34 | 503 | 776 |
| s641 | 7 | 11 | 261 | 321 |
| s713 | 7 | 11 | 284 | 321 |
| s820 | 0 | 32 | 95 | 559 |
| s832 | 2 | 33 | 146 | 575 |
| s838.1 | 78 | 120 | 3246 | 4749 |
| s953 | 5 | 50 | 159 | 847 |
| s1196 | 7 | 20 | 198 | 413 |
| s1238 | 7 | 21 | 198 | 431 |
| s1423 | 0 | 5 | 184 | 207 |
| s5378 | 22 | 31 | 759 | 883 |
| s9234 | 216 | 237 | 11766 | 12772 |
| s13207 | 171 | 179 | 10796 | 11101 |
| s15850 | 237 | 246 | 11826 | 12267 |
| s38417 | 658 | 795 | 71491 | 85813 |
| s38584 | 187 | 195 | 11529 | 12077 |

Table 9: Number of deterministic patterns and storage requirements for the complete test data (in bits).

The results show that an optimized random test in fact considerably reduces the number of deterministic patterns and the overall test data storage. This is particularly true for the circuits known as random pattern resistant. E.g. for circuit c7552 the number of deterministic patterns is reduced from 92 to 51 and the reduction in test data storage is about 5K. For circuit s38417 the best scheme eliminates 137 deterministic patterns, which leads to a reduction in test data storage of more than 14K. As shown in Table 10 already with standard ATPG the proposed technique requires less test data storage than an approach based on storing a compact test set (cf. [16, 18, 22, 27]).

| Circuit | Deterministic patterns | | Test data storage (bits) | |
|---|---|---|---|---|
| | Best scheme | Compact Test Set | Best scheme | Compact Test Set |
| c2670 | 73 | 51 | 4186 | 8007 |
| c3540 | 1 | 97 | 59 | 4850 |
| c7552 | 51 | 84 | 6889 | 17304 |
| s420.1 | 22 | 43 | 503 | 1505 |
| s641 | 7 | 24 | 261 | 1296 |
| s713 | 7 | 23 | 284 | 1242 |
| s820 | 0 | 95 | 95 | 2185 |
| s832 | 2 | 96 | 146 | 2208 |
| s838.1 | 78 | 75 | 3246 | 5025 |
| s953 | 5 | 77 | 159 | 3465 |
| s1196 | 7 | 117 | 198 | 3744 |
| s1238 | 7 | 129 | 198 | 4128 |
| s1423 | 0 | 29 | 184 | 2639 |
| s5378 | 22 | 104 | 759 | 22256 |
| s9234 | 216 | 116 | 11766 | 28652 |
| s13207 | 171 | 235 | 10796 | 164500 |
| s15850 | 237 | 113 | 11826 | 69043 |
| s38417 | 658 | 91 | 71491 | 151424 |
| s38584 | 187 | 141 | 11529 | 206424 |

Table 10: Amount of test data storage for the proposed approach and for storing a compact test set.

It can be expected, that the test data storage for the presented approach could be reduced even further, if an ATPG tool specially tailored for the encoding scheme were used as described in [15].

## 7 Conclusion

A scheme for generating mixed-mode test programs for embedded processors has been presented. The test program uses both new, highly efficient random test schemes and a new software-based encoding of deterministic patterns.

It has been shown that the careful selection of primitive polynomials for LFSR-based random pattern generation has a strong impact on the number of undetected faults, and a multiple-polynomial random pattern scheme provides significantly better results in many cases. The quality of the random scheme has the main impact on the overall size of a mixed-mode test program. As an example, for the processor INTEL 80960CA test programs were generated, and for all the benchmark circuits a complete coverage of all non-redundant faults was obtained.

## 8 References

1 S. B. Akers, W. Jansz: Test Set Embedding in a Built-in Self-Test Environment; Proc. IEEE Int. Test Conf., Washington D.C., 1989, pp. 257-263

2 P. Bardell, W. H. McAnney, J. Savir: Built-in Test for VLSI; Wiley-Interscience, New York, 1987

3 Z. Barzilai, D. Coppersmith, A. L. Rosenberg: Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing; IEEE Trans. on Comp., Vol. C-32, No. 2, February 1983, pp. 190-194

4 F. Brglez and H. Fujiwara: A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran; IEEE Int. Symp. on Circuits and Systems, Kyoto, 1985

5 F. Brglez et al.: Hardware-Based Weighted Random Pattern Generation for Boundary-Scan; Proc. IEEE Int. Test Conf., Washington D.C., 1989, pp. 264 - 274

6 F. Brglez, D. Bryan and K. Kozminski: Combinational Profiles of Sequential Benchmark Circuits; Proc. IEEE Int. Symp. on Circuits and Systems, 1989, pp. 1929-1934

7 M. Chatterjee and D. K. Pradhan: A New Pattern Biasing Technique for BIST; Proc. of VLSI Test Symp., 1995, pp. 417-425

8 C. Dufaza, H. Viallon, C. Chevalier: BIST Hardware Generator for Mixed Test Scheme; Proc. Europ. Design and Test Conf., Paris, 1995

9 E.B. Eichelberger and E. Lindbloom: Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test, IBM Journal of Research and Development, Vol. 27, No. 3, May 1983

10 A. Flint: Multichip Module Self-Test Provides Means to Test at Speed; EE-Evaluation Engineering, pp. 46-55, September 1995

11 S. W. Golomb: Shift Register Sequences; Holden-Day, San Francisco, 1967

12 S. Gupta, J. Rajski, J. Tyszer: Test Pattern Generation Based On Arithmetic Operations; Proc. Int. Conf. on Computer-Aided Design, San Jose, Ca., 1994, pp. 117-124

13 S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois: Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers; Proc. IEEE Int. Test Conf., Baltimore 1992, pp. 120-129

14 S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois: Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers; IEEE Trans. on Comp., Vol. 44, No. 2, Feb. 1995, pp. 223-233

15 S. Hellebrand, B. Reeb, S. Tarnick, H.-J. Wunderlich: Pattern Generation for a Deterministic BIST Scheme; Proc. IEEE/ACM Int. Conf. on CAD-95, San Jose, Ca., Nov. 1995

16 H. Higuchi, N. Ishiura, and S. Yajima: "Compaction of Test Sets Based on Symbolic Fault Simulation", Synthesis and Simulation Meeting and Int. Interchange, pp. 253-262, 1992

17 P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, H. C. Card: Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test; IEEE Trans. on CAD, pp. 842-859, Aug. 1989

18 S. Kajihara, I. Pomeranz, K. Kinoshita, S. M. Reddy: "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits", Proc. 30th ACM/IEEE Design Automation Conf., 1993, pp. 102-106

19 B. Koenemann: LFSR-Coded Test Patterns for Scan Designs; Proc. Europ. Test Conf., Munich, 1991, pp. 237-242

20 B. Koenemann, J. Mucha, G. Zwiehoff, Built-In Logic Block Observation Techniques, Proc. Test Conf., Cherry Hill, NJ, 1979, pp. 37-41

21 N. Mukherjee, M. Kassab, J. Rajski, J. Tyszer: Accumulator Built-In Self Test for High-Level Synthesis; VLSI Test Symp., 1995, pp. 132-139

22 L.N. Reddy, I. Pomeranz, and S.M. Reddy: "ROTCO: A Reverse Order Test Compaction Technique", Proc. IEEE EURO-ASIC Conf., September 1992, pp. 189-194

23 J. Savir, W.H. McAnney: A Multiple Seed Linear Feedback Shift Register; IEEE Trans. on Comp., pp. 250-252, Feb. 1992

24 M. Schulz and E. Auth: Advanced Automatic Test Generation and Redundancy Identification Techniques; Proc. 18th Int. Symp. on Fault-Tolerant Computing, Tokyo 1988, pp. 30-35

25 A. P. Stroele: A Self-Test Approach Using Accumulators as Test Pattern Generators; Proc. Int. Symp. on Circuits and Systems, 1995, pp. 2120-2123

26 N. A. Touba and E. J. McCluskey: Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST; Proc. IEEE Int. Test Conf., Washington, D.C., pp. 674-682, 1995.

27 G. Tromp: Minimal Test Sets for Combinational Circuits; Proc. IEEE Int. Test Conf., 1991, pp. 204-209

28 I. Voyiatzis, A. Paschalis, D. Nikolos, C. Halatsis: Accumulator-Based BIST Approach for Stuck-Open and Delay Fault Testing; Proc. Europ. Design & Test Conf., 1995, pp. 431-435

29 E. J. McCluskey, L. T. Wang: Circuits for Pseudo-Exhaustive Test Pattern Generation; Proc. IEEE Int. Test Conf.; 1986, pp. 25-37

30 H.-J. Wunderlich: Self Test Using Unequiprobable Random Patterns; Proc. IEEE 17th Int. Symp. on Fault-Tolerant Computing, FTCS-17, Pittsburgh 1987, pp. 258-263

31 H.-J. Wunderlich: Multiple Distributions for Biased Random Test Patterns; Proc. IEEE Int. Test Conf., Washington D.C., 1988, pp. 236-244

32 N. Zacharia, J. Rajski, J. Tyszer: Decompression of Test Data Using Variable-Length Seed LFSRs; Proc. 13th VLSI Test Symp., pp. 426-433, 1995