Accumulator Based Deterministic BIST

Rainer Dorsch, Hans-Joachim Wunderlich

Computer Architecture Lab
University of Stuttgart

ABSTRACT— *Most built-in self test (BIST) solutions require specialized test pattern generation hardware which may introduce significant area overhead and performance degradation. Recently, some authors proposed test pattern generation on chip by means of functional units also used in system mode like adders or multipliers. These schemes generate pseudo-random or pseudo-exhaustive patterns for serial or parallel BIST. If the circuit under test contains random pattern resistant faults a deterministic test pattern generator is necessary to obtain complete fault coverage.*

In this paper it is shown that a deterministic test set can be encoded as initial values of an accumulator based structure, and all testable faults can be detected within a given test length by carefully selecting the seeds of the accumulator. A ROM is added for storing the seeds, and the control logic of the accumulator is modified. In most cases the size of the ROM is less than the size required by traditional LFSR-based reseeding approaches.

KEYWORDS— *BIST, hardware pattern generator, embedded cores*

I. Introduction

The complexity of systems-on-chip makes Built-In Self Test (BIST) an favorable method for system testing [1], [2], [3]. The testability of embedded cores may suffer from limited accessibility, and necessary test information is often hidden in order to protect intellectual property (IP). This kind of problem is solved if the system is equipped with BIST features, and the functionality of the core may be used not only for implementing the system mode, but also for test pattern generation and test response evaluation. Accumulator based structures may work in an autonomous mode for generating patterns with some pseudo-random or pseudo-exhaustive properties (see fig. 1) or compress test data like an LFSR during signature analysis.

The advantages of this approach are twofold: as a specialized BIST circuitry is not needed, the hardware overhead is reduced to some modifications for implementing BIST control, and since BIST circuitry within the data path is completely avoided, this BIST method will not affect system performance.

This work was supported by the DFG under grant Wu 245/1-1.

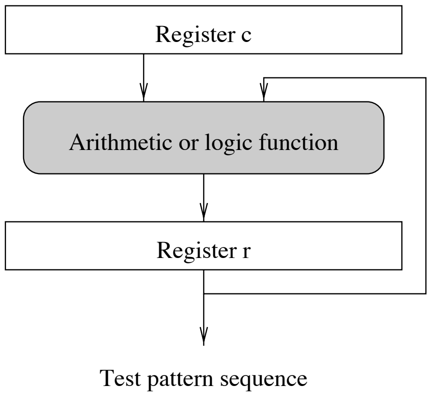


Fig. 1. A typical accumulator structure used as test pattern generator. In each cycle the constant value of register c is added to register r. The content of register r is a test pattern.

The use of accumulator based structures for test response compaction has widely been investigated in literature [4], [5], [6], and the aliasing probabilities of these structures have the same magnitude as the aliasing probabilities of the LFSR-based signature analysis.

Test pattern generation may be performed by using a variety of functional units in the accumulator based structure of fig. 1. Investigations are known about properties of test patterns generated by simple adders [7], [8], ones- and twos-complemented subtractors [6], [9], and more complex multipliers and MAC circuits [10]. All of them may generate pseudo-exhaustive or pseudo-random patterns with a similar quality as LFSRs do, and may reach a comparable fault coverage.

For random pattern testable circuits, the functional BIST approach may work efficiently, but it fails, if random pattern resistant faults are present in the circuits. Random pattern resistant faults may be removed by test point insertion [11], [12], which cause some hardware overhead and additional delays in the signal path. Moreover, test point insertion requires the availability of a structural representation of the core to be modified.

Recently some deterministic BIST schemes have been developed, like Bit-Flipping [13], [14], Bit-Fixing [15], [16], Pattern-Mapping [17], or Reseeding [18], [19], [20], which use dedicated hardware to generate deterministic test patterns. To feed the patterns to the circuit under test (CUT) these solutions have to add multiplexors to the signal path. While these methods may obtain complete fault coverage, they require considerable hardware overhead and may cause some performance degradation.

Until now, system hardware based BIST schemes which can test random-pattern-resistant faults, were restricted to the use of processors present in the system [21]. In this paper, we show how to combine the advantages of deterministic BIST and of BIST schemes based on functional units. We are developing a novel scheme, that is used to compute seeds for an accumulator with a simple adder, which covers a complete deterministic test set. These structures are present in many data-path architectures and in specialized digital signal processing circuits.

The use of an accumulator as test pattern generator is described in section II. In section III an algorithm is outlined for the symbolical search for optimal seeds. We present heuristics to reduce the time complexity of this algorithm in section IV. The use of fault simulation to improve the suboptimal solution is discussed in section V. Since to the best of our knowledge this is the first time an adder based accumulator is used as a deterministic test pattern generator, we demonstrate the feasibility of our solution by applying it to the random pattern resistant ISCAS89 and ISCAS85 benchmarks in section VI.

II. Accumulator Based Pattern Generation

The typical accumulator structure of fig. 2 serves as a test pattern generator with an adder as arithmetic unit. The test patterns are given by

$$r_i = (r_{i-1} + c) \bmod 2^N = (r_0 + i \cdot c) \bmod 2^N, \qquad (1)$$

where r_i is generated at clock cycle i if the accumulator is initialized with r_0, and the constant c is added iteratively. The pair (r_0, c) is called a seed, and for each deterministic test set $T \subset \{0,1\}^N$ there are a seed and a number n so that $T \subset \{r_i | 0 \leq i \leq n\}$ and all patterns of T are generated by the accumulator within n clock cycles. For $c = 1$ and $n = 2^N$ the accumulator structure works like a counter, and every pattern can be generated. The problem is to find a seed (r_0, c) so that a minimum n satisfies $T \subset \{r_i | 0 \leq i \leq n\}$.

Example 1: Assume a five input circuit with random pattern resistant faults, which requires the deterministic test set

$$T = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & - & 0 & 1 \\ 0 & 1 & 1 & 1 & - \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad (2)$$

We allow unspecified bits "-" in a test pattern so that each test cube t_i may represent a set of patterns, of which one must be generated. With the seed $(11101, 00101)$, the deterministic T is generated within $n = 16$ clock cycles (see fig. 2).

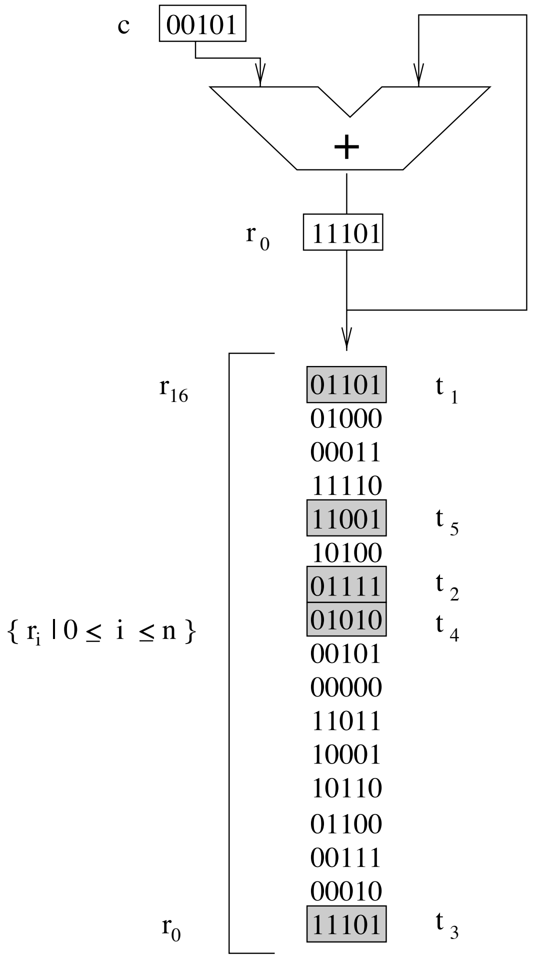


Fig. 2. Embedding a deterministic test set T (highlighted) in an accumulator sequence $\{r_i | 0 \leq i \leq n\}$. The accumulator register contains its initial content. The topmost pattern in the pattern sequence is generated last.

To generate all patterns of T, a single optimal seed may require a large number of clock cycles, which may increase exponentially with the number of bits per pattern.

Hence, the deterministic test set $T = T_1 \cup T_2 \cup \cdots \cup T_w$ is partitioned into w subsets with w different seeds $s^1 :=$

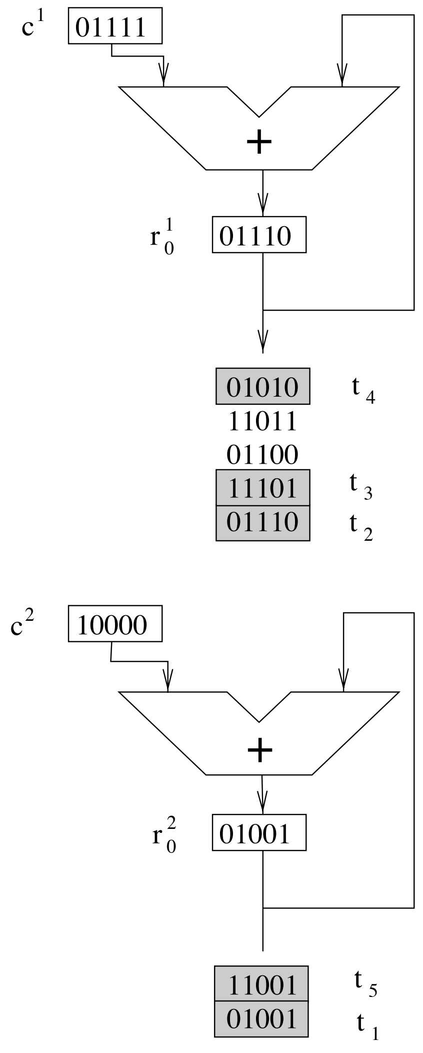


Fig. 3. Reseeding of accumulators

$(r_0^1, c^1), \ldots, s^w := (r_0^w, c^w)$. Each of the seeds s^i starts a run of n_i clock cycles. We should minimize w with the restriction

$$\sum_{i=1}^{w} n_i \leq n = \text{const.}, \tag{3}$$

i.e. the total number of applied test patterns is restricted. The number of seeds w corresponds directly to the storage effort.

Example 2: In the previous example $n = 16$ clock cycles have been required. This number can be reduced by partitioning the test set into $T_1 = \{t_2, t_3, t_4\}$ and $T_2 = \{t_1, t_5\}$. The seed $s_1 = (01110, 01111)$ generates T_1 within $n_1 = 4$ cycles, and seed $s_2 = (10000, 01001)$ generated T_2 using $n_2 = 1$ cycle (see fig. 3).
In this very small example we have to store two seeds for encoding 5 patterns, for larger test sets the reseeding approach is much more efficient. In a certain way the adder based reseeding corresponds to the reseeding of LFSRs as proposed by Koenemann[22].

III. Symbolic Search for Single Seed Solutions

In this section, an optimal solution is discussed for constructing a single seed $s := (r_0, c)$ that generates a deterministic test set T in a minimal number n of cycles. The optimal solution will rarely be computable and will require large test times, but it is the basis for heuristics of the suboptimal solution presented in section IV.

Let

$$s_j^m := \{(r_0, c) | t_j = (r_0 + m \cdot c) \bmod 2^N\} \tag{4}$$

be the set of seeds which generates test pattern t_j in cycle m. The set

$$\bigcup_{m=1}^{n} s_j^m \tag{5}$$

is the set of all seeds covering test pattern t_j at least once in n cycles. As the entire test set $T = \{t_1, \ldots, t_k\}$ has to be generated we are interested in the set of seeds

$$o_n = \bigcap_{j=1}^{k} \left(\bigcup_{m=0}^{n} s_j^m \right). \tag{6}$$

Each seed of o_n covers all patterns within n cycles. Let $n_{\text{opt}} := \min\{n | o_n \neq= emptyset\}$ be the smallest number of cycles so that all the patterns can be generated by a single seed. All seeds of $o_{n_{\text{opt}}}$ are optimal.

The sets o_n consist of boolean vectors (r_0, c) of length $2N$. Such a set is efficiently described by its characteristic function

$$\chi_{o_n} : \{0,1\}^{2N} \to \{0,1\} \text{ with } \chi_{o_n}(r,c) = 1 - \ (r,c) \, \epsilon \, o_n. \tag{7}$$

Let χ_j^m be the characteristic function of the set s_j^m mentioned in eqn. (6). Then we have

$$\chi_{o_n} = \chi_{\bigcap_{j=1}^{k} \left(\bigcup_{m=0}^{n} s_j^m \right)} = \prod_{j=1}^{k} \sum_{m=0}^{n} \chi_j^m. \tag{8}$$

The evaluation of eqn. (8) requires an efficient way for manipulating boolean formulas, and for this purpose we use *reduced ordered binary decision diagrams* (ROBDD) [24]. First we describe how to compute the characteristic function χ_j^m of the set of seeds

$$s_j^m = \{(r_0, c) | t_j = (r_0 + m \cdot c) \bmod 2^N\} \tag{9}$$

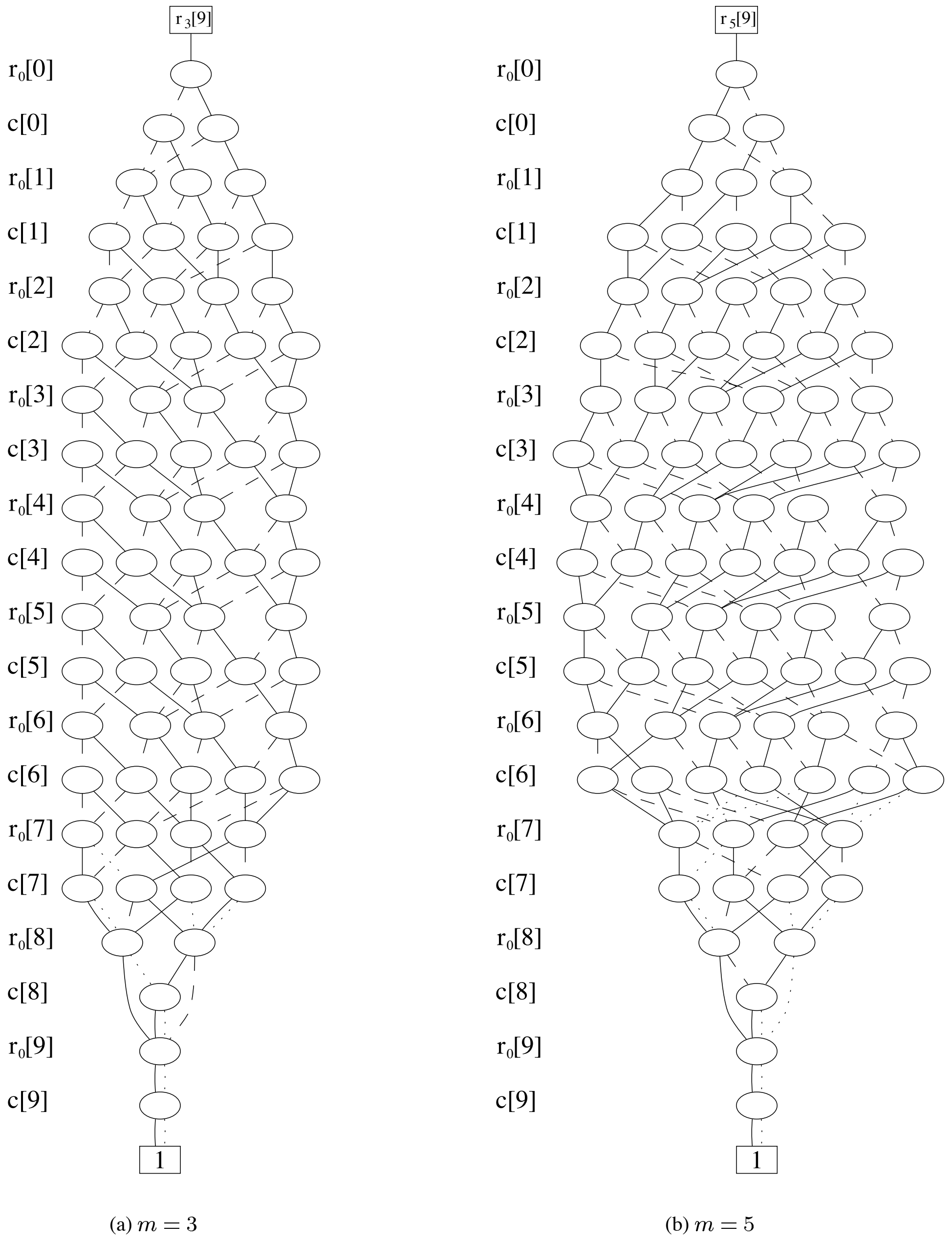


Fig. 4. Examples for the regularity of ROBDDs for $r_m[9](r_0, c)$. All nodes on the same level correspond to the same variable, of which the name is shown on the left of each diagram. Dotted lines indicate complemented, dashed lines indicate regular "else" arcs [23].

by generating at most N ROBDDs, one ROBDD $\chi^m_{j,i}$ for each specified bit $t_j[i]$ of the pattern t_j. The i^{th} bit of

$$r_m = (r_0 + m \cdot c) \bmod 2^N \tag{10}$$

is a boolean function $r_m[i](r_0, c)$. The ROBDD of $r_m[i](r_0, c)$ can be computed by applying the ROBDD of a full adder repeatedly [25]. The size of the ROBDD of $r_m[i](r_0, c)$ increases linearly in both the number cycles m and bit position i (see fig. 4) for the variable ordering $(r_0[0], c[0], r_0[1], c[1], \ldots, r_0[N-1], c[N-1])$.

For each specified bit in t_j, we must have $r_m[i](r_0, c) = t_j$, and the characteristic function of each specified bit is

$$\chi^m_{j,i}(r_0, c) = \begin{cases} r_m[i](r_0, c) & \text{for} \quad t_j[i] = 1 \\ \neg r_m[i](r_0, c) & \text{for} \quad t_j[i] = 0 \\ 1 & \text{for} \quad t_j[i] = - \end{cases} \tag{11}$$

and the characteristic function of the set of seeds generating t_j in cycle m is given by

$$\chi_j^m(r_0, c) = \prod_{i=1}^{N-1} \chi_{j,i}^m(r_0, c). \qquad (12)$$

By applying eqn. (8) to the functions χ_j^m, we obtain the set of all seeds by the characteristic function

$$\chi_{o_n} = \prod_{j=1}^{k} \sum_{m=0}^{n} \prod_{i=1}^{N} \chi_{j,i}^m \qquad (13)$$

Example 3: Assume, we have to generate the test pattern $t_1 = (01-01)$ in clock cycle $m = 1$. The corresponding set of seeds s_1^1 is expressed by its characteristic function $\chi_1^1(r_0, c)$ in a ROBDD form. The seeds in s_1^1 must satisfy

$$\begin{aligned} r_1[4](r_0,c) &= t_1[4] = 0 \\ r_1[3](r_0,c) &= t_1[3] = 1 \\ r_1[1](r_0,c) &= t_1[1] = 0 \\ r_1[0](r_0,c) &= t_1[0] = 1. \end{aligned} \qquad (14)$$

Hence the charcteristic function of the set of seeds must satisfy the following formulas

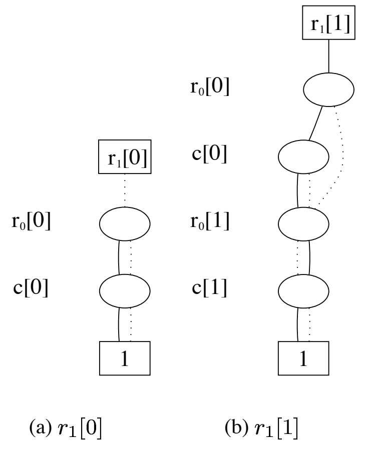
$$\begin{aligned} \chi_{1,0}^1 &= r_1[0] \\ \chi_{1,1}^1 &= \neg r_1[1] \\ \chi_{1,2}^1 &= 1 \\ \chi_{1,3}^1 &= r_1[3] \\ \chi_{1,4}^1 &= \neg r_1[4]. \end{aligned} \qquad (15)$$

We show the corresponding four BDDs in fig. 5. The characteristic function χ_1^1 for the pattern t_1, we are looking for, is according to eqn. (12) given by

$$\chi_1^1(r_0, c) = \prod_{i=0}^{4} \chi_{1,i}^1. \qquad (16)$$

We show the corresponding ROBDD in fig. 6.

Unfortunately, the direct application of eqn. (13) is not always possible for two reasons: The smallest n with $o_n \neq \emptyset$ (equivalent to $\chi_{o_n} \not\equiv 0$) may still be too large for any practical application, and computing χ_{o_n} requires complex ROBDD operations, which may not be feasible for practical examples. n_{opt} may be reduced by partitioning the test set as outlined in section II. In the next section, heuristics for generating suboptimal solutions are discussed.



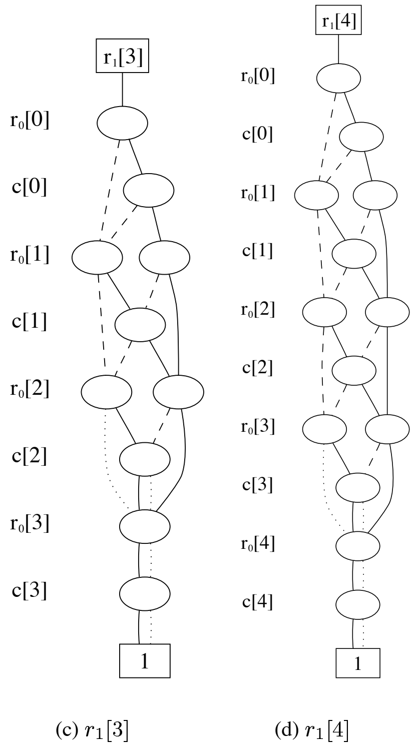


Fig. 5. ROBDDs for the zeroth (a), first (b), third (c), and fourth (d) bit of an adder. The second bit in $(10-10)$ is not specified, therefore it is not necessary to generate a ROBDD for the second adder bit $r_1[2]$.

IV. Hybrid Search

Although efficient ROBDD representations for χ_j^m exist, the optimal solution χ_{o_i} (eqn. (8)) may not be efficiently represented by ROBDDs. The ROBDD explosion problem can be avoided by approximations. We are proposing an algorithm for generating suboptimal solutions, which is still good in terms of test length, but we might need additional seeds to obtain 100% fault efficiency. To get a feasible sub-

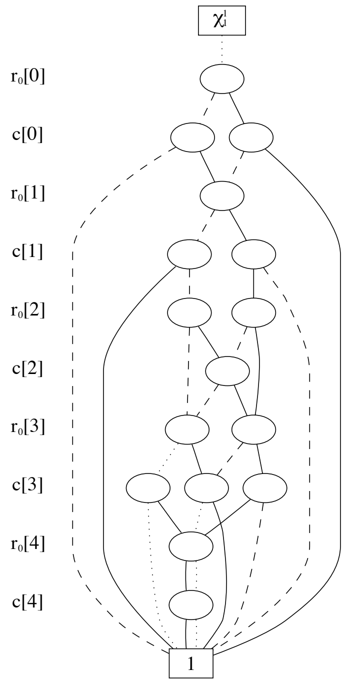


Fig. 6. The ROBDD of the characteristic function r_1 of the set of seeds, which generates pattern $t_1 = (01-01)$ in iteration $m = 1$.

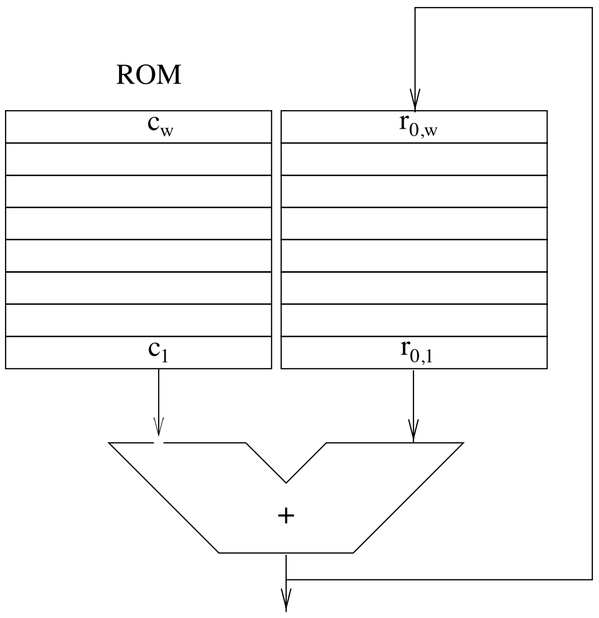


Fig. 7. Partitioning a seed by using a small adder and a register file.

optimal solution, two heuristics are applied.

The first heuristic partitions the set of N variables into smaller blocks of size B, so that $w := \lceil N/B \rceil$ additions of B-bit words have to be performed instead of one addition of an N-bit word. The second heuristic reduces the search space.

A. Partitioning of variables

For a large scan path with several thousands of flipflops, the word size of the adder in fig. 1 is neither realistic nor computationally tractable. Usually, adders are found up to a word size of $B = 64$ bits, and the pattern generation may be performed either by using w different adders or by distributing w additions over w clock cycles. In the latter case, a register file is required for storing w different accumulator values $r_{i,1}, r_{i,2}, \ldots, r_{i,w}$ and a ROM, which stores the constants $c_1, c_2, \ldots, c_w$ (see fig. 7). In any case, there is no need to propagate a carry between different blocks, and the variables of different blocks are separated.

For $h := 1, \ldots, w$ let

$$\chi_j^{m,h} = \prod_{i=(h-1)\,B}^{h\,B-1} \chi_{j,i-(h-1)\cdot B}^{m} \qquad (17)$$

be the characteristic function for the set of seeds $(c_h, r_{0,h})$ of the h^{th} block. This is a function in $2B$ variables instead of $2N$ variables, and the optimal set of seeds for partitioned variables is described by

$$\chi_{o_n} = \prod_{j=1}^{k} \sum_{m=0}^{n} \prod_{h=1}^{w} \chi_j^{m,h} \qquad (18)$$

The expression above is still expensive to compute as the sum operation may cause an explosion of the ROBDD representing χ_{o_n}.

B. Reducing the search space

So far we allowed a test pattern to be generated in any cycle $m = 0 \ldots n$, which leads to an expensive sum operation. If exactly one cycle $m_j \epsilon \{0, \ldots, n\}$ is assigned to a pattern t_j eqn. (18) is reduced to

$$\chi_{o_n} = \prod_{j=1}^{k} \prod_{h=1}^{w} \chi_j^{m_j,h} = \prod_{h=1}^{w} \prod_{j=1}^{k} \chi_j^{m_j,h} \qquad (19)$$

Now each block can be dealt with independently.

Due to preselecting cycle m_j, in which each test pattern t_j is generated, we may loose solutions, and m_j must be fixed carefully. The preselection is controlled by a cost function $c(t_j)$ for generating t_j by a seed of o in a given cycle. We used the cost function

$$c(t_j) = \frac{\text{Expected remaining cardinality of } o_n}{\text{Actual remaining of cardinality of } o_n} \qquad (20)$$

The "remaining cardinality" is defined by

$$\frac{\text{Cardinality of } o_n \text{ with } t_j}{\text{Cardinality of } o_n \text{ w/o } t_j} \qquad (21)$$

Each specified bit reduces the cardinality of o_n in average by a factor of two. Thus, the expected remaining cardinality for a pattern t_j is given by $1/2^{x_j}$ where x_j is the number of specified bits in t_j. The actual remaining cardinality of o_n is given by

$$\frac{\mathrm{mit}(\chi_o(\text{with } t_j))}{\mathrm{mit}(\chi_o(\text{w/o } t_j))}, \qquad (22)$$

where $\mathrm{mit}(\chi_o)$ is the number of minterms of χ_o. This yields our cost function

$$c(t_j) = \frac{\mathrm{mit}(\chi_o(\text{w/o } t_j))/\mathrm{mit}(\chi_o(\text{with } t_j))}{2^{x_j}}. \qquad (23)$$

Now the test cycles are assigned, iteratively.

First we take two test patterns t_s and t_t with a maximum number of specified bits and set $m_s = 0$ and $m_t = 1$. Next we compute for each remaining pattern t_u the number of possible seeds (that is the number of minterms of χ_o) if we set m_u=2 and recompute χ_o. We choose pattern $t_{\min}$ with $m_{\min}$=2, which has the least costs. We iterate now for $m_j = 3, 4, 5, \ldots$ until all patterns are generated.

It eventually happens that the seed set o_n is empty for all possible patterns t_j. In this case, we store a seed from the previous χ_o and iterate the complete procedure. The algorithm is written more formally in a C-like syntax in algorithm 1.

Algorithm 1 Computation of m_j

```
$T$ = set of test patterns
$\chi_o = 1$
while $T \neq \emptyset$ do
  Select two patterns $t_1 \epsilon T$ and $t_2 \epsilon T$
  $m_1 = 0$
  $m_2 = 1$
  $\chi_o = \chi_{t_1}^0 \cdot \chi_{t_2}^1$
  $T = T \backslash \{t_1, t_2\}$
  $cycle = 2$
  repeat
    $\chi_o^{\text{best}} = 0$
    for $t_j \epsilon T$ do
      $\chi_o^{\text{current}} = \chi_o \cdot \chi_{t_j}^{cycle}$
      $c = \text{cost}(\chi_o^{\text{current}})$
      if c is minimal then
        $\chi_o^{\text{best}} = \chi_o^{\text{current}}$
        $m_j = cycle$
        $t_{\min} = t_j$
      end if
    end for
    if $\chi_o^{\text{best}} \neq \emptyset$ then
      $\chi_o = \chi_o^{\text{best}}$
      $T = T \backslash t_{\min}$
    end if
    $cycle++$
  until $\chi_o^{\text{best}} = 0$
  store one seed of $\chi_o$
end while
```

C. Reducing the Number of Seeds

A common method to reduce the storage space for patterns or seeds is to allow intermediate patterns, which do not necessarily find new faults. Assume, we allow n patterns to be applied in total, and $k' \leq k$ deterministic test patterns must still be generated in $n' \leq n$ cycles. We limit the number of useless patterns by generating a new pattern within the next

$$2 \cdot \frac{n' - k' + 1}{k'} \qquad (24)$$

cycles, which we call an *iteration*. The number of intermediate useless patterns is limited. If a new patterns t_j is to be selected, the corresponding cycle m_j is within these cycles, and the best time slot is selected by the cost function $c(t_j)$ as described in section IV-B.

Example 4: In this example, we demonstrate, how the proposed algorithm encodes the patterns

$$T = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & - & 0 & 1 \\ 0 & 1 & 1 & 1 & - \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad (25)$$

given in eqn. (2). The first two selected patterns may be 0111− and 11101. Two seeds generate these two patterns in cycle 0 and 1: $(01110, 01111)$ and $(01111, 01110)$. For the next pattern we compute the cost function for $2 \cdot (10 - 2 - 2)/3 = 4$ cycles (see eqn. (24)). We give the number of minterms and the cost function in brackets.

cycle	01010	01-01	11001
2	$0(\infty)$	$0(\infty)$	$0(\infty)$
3	$0(\infty)$	$0(\infty)$	$1(1/32)$
4	$1(1/32)$	$0(\infty)$	$0(\infty)$
5	$0(\infty)$	$0(\infty)$	$1(1/32)$

We find that two patterns have the same costs. We choose 11001 because it is generated in a earlier cycle. The seed $(01111, 01110)$ generates all three patterns 11101, 0111−, and 11001 within $n = 3$ cycles.
We repeat the same procedure for the two remaining patterns $01010, 01 - 01)$. It turns out that in cycles 4 to 8 none of the two patterns can be generated by the remaining seed. Therefore we store the obtained seed $(01111, 01110)$ and $n_1 = 3$.

The remaining two patterns are generated by the seed $(01010, 11111)$ in one cycle ($n_2 = 1$).
Note, in real circuits, the number of unspecified bits and thus the cardinality of o_n is much larger than in this example.

D. *Controlling ROBDD Growth*

During the process of pattern selection and pattern encoding described in section IV-B and IV-C, the ROBDDs and therefore the computation time may increase. We implemented a trade-off between the time efficiency of the program and the encoding efficiency of the results by a simple time limit strategy: We give a time limit for the complete program and observe the remaining time, which is the time limit minus the time the computations already runs. If the product of the time for finding m_j and the number of remaining test patterns is larger than the remaining time, we stop the computation and take the pattern and cycle with the best costs in the current iteration.

V. Fault Simulation

The deterministic test set T may be reduced by intertwining pattern encoding and fault simulation. Each time when a seed (r_0, c) is determined, all patterns generated by this seed are simulated, the detected faults are removed from the fault list, and the corresponding test patterns are not encoded in future steps of the algorithm.

Moreover, we estimate the probability, that a fault is detected by random patterns, by fault simulation. We try to encode only patterns which are not covered by n' random patterns in average at least twice. The test itself does not contain any random pattern application.

VI. Experiments

To evaluate our deterministic test pattern generator, we performed simulations for the ISCAS85 [26] and ISCAS89 benchmarks [27], which are not randomly testable by 10000 test patterns. The test length was 10000 patterns, and the computation time was restricted to 24 hours. We estimated the fault detection probability by fault simulation of 10000 patterns. The encoding was performed by using an adder of the size of $B = 16$ bits. We compare our results with the storage effort of a compact test set [28], [29], [30], [31] and the storage requirements for the seeds of a test pattern generator which needs a complete processor [21], in table I.

A seed (r_0, c) of the adder based approach always consists of two patterns, hence all the entries of the corresponding column are even. The adder based approach needs always less patterns than the compact test. These patterns were selected from the optimum of different published references [28], [29], [30], [31]. Especially for the larger circuits, the presented approach also outperforms a processor based encoding scheme [21] while only a simple adder based accumulator is used.

circuit	PPIs	Deterministic patterns		
		Adder	C-Test	Processor
s420.1	34	14	43	22
s641	54	12	24	7
s713	54	10	23	7
s820	23	16	95	0
s832	23	16	96	2
s838.1	66	52	75	78
s953	45	6	77	5
s1196	32	10	117	7
s1238	32	10	129	7
s1423	91	6	29	0
s5378	214	16	104	22
s9234	247	30	116	216
s13207	700	18	235	171
s15850	611	26	113	237
s38417	1664	44	91	658
s38584	1464	30	141	187
c2670	233	36	51	73
c7552	207	46	97	51

TABLE I. Number of test patterns to store for the proposed method, for compact test sets and for a processor based approach.

To evaluate the dependence of the number of patterns to store on the test length, we performed an additional simulation with 5000 patterns. Table II shows the data for a test length of 10000 patterns in column 2 and 3 and for a test length of 5000 patterns in column 4 and 5. We have to store $2\,w$ patterns. This number increases slightly for most benchmarks, whereas the computation time t in hours for the seeds decreases if the test length is reduced. The computation time is basically determined by the number of generated patterns per seed and the total number of deterministic patterns.

The processor based scheme performs an additional compression of patterns by exploiting unspecified bits. But the seeds generated by the presented method contain unspecified bits, as well. Table III shows the number of unspecified and specified bits of the stored seeds, and compares these numbers to the number of bits to be stored by the processor based approach. In some cases, storing complete seeds requires less memory than the data requirements for the processor. The specified part of the seed is distinctively less than the processor requirements, and can be used for further optimizations.

Overall, the experiments show the following trends:

- For small circuits (less than 20 inputs or scan elements), it is favorable to try to get the optimal solution, or if this does not work to divide the patterns into groups and find the optimal solution for them.

circuit	10000 patterns		5000 patterns	
	t/h	$2\,w$	t/h	$2\,w$
s420.1	0.2	14	0.1	18
s641	0.5	12	0.4	6
s713	0.5	10	0.2	6
s820	1.2	16	0.4	14
s832	1.0	16	0.2	14
s838.1	0.8	52	0.6	52
s953	0.4	6	0.2	6
s1196	0.9	10	0.4	20
s1238	0.7	10	0.2	20
s1423	20.1	6	5.1	8
s5378	17.4	16	3.2	18
s9234	23.7	30	8.8	38
s13207	20.5	18	6.7	22
s15850	21.1	26	3.3	42
s38417	23.8	44	14.8	44
s38584	20.2	30	10.0	46
c2670	2.5	36	5.4	36
c7552	7.9	46	5.6	52

TABLE II. Dependence of the number $2\,w$ of patterns to store and computation times for the seeds on the test length for the proposed method.

circuit	Bits of seeds	DC bits	Spec. bits	Proc.
s420.1	476	121	355	503
s641	648	201	447	261
s713	540	217	323	284
s820	368	13	355	95
s832	368	7	361	146
s838.1	3432	1681	1751	3246
s953	270	71	199	159
s1196	320	39	281	198
s1238	320	79	241	198
s1423	546	52	294	184
s5378	3424	1964	1460	759
s9234	7410	1538	5872	11766
s13207	12600	5766	6834	10796
s15850	15886	10368	5518	11826
s38417	73216	44052	29164	71491
s38584	43920	36955	6965	11529
c2670	8388	5980	2408	4178
c7552	9522	3110	6412	6889

TABLE III. Unspecified bits in the seeds allow further improval of the encoding scheme.

- For mid-size and large circuits, the presented heuristics are satisfactory, although a sophisticated tuning of the heuristics to determine the pattern sequence may improve the results.
- Large circuits show a linear growth of memory consumption and sublinear growth of computation time, given that the number of test patterns for hard faults stays constant.

VII. Conclusions

BIST schemes, which exploit functional units of the system for test pattern generation, can be efficiently extended to a deterministic BIST strategy. For adder-based accumulator structures, a few seeds can be computed so that the output sequence contains a complete deterministic test set. The storage requirements for the seeds are competitive to known reseeding approaches, but hardware overhead and performance impact of the new approach is less than it is for the standard BIST schemes.

VIII. Acknowledgements

We used the *Colorado University Decision Diagram* (CUDD) package [32] to perform the ROBDD computations. We would like to acknowledge this excellent work, which shows good performance and flexibility and is well documented. Further we would like to thank Fabio Somenzi for his support in CUDD related questions during the development of our code.

References

[1] C. Maunder, “A Universal Framework for Managed Built-In Test,” in *ITC*, pp. 21–29, IEEE, 1993.

[2] R. Chandramouli and S. Pateras, “Testing Systems on a Chip,” *IEEE Spectrum*, pp. 42–47, November 1996.

[3] M. Hunt and J. A. Rowson, “Blocking in a System on a Chip,” *IEEE Spectrum*, pp. 35–41, November 1996.

[4] J. Rajski and J. Tyszer, “Recursive Pseudoexhaustive Test Pattern Generation,” in *ToC*, pp. 1517–1521, IEEE, 1993.

[5] J. Rajski and J. Tyszer, “Accumulator-Based Compaction of Test Responses,” *IEEE Transactions on Computers*, vol. 42, pp. 643–650, June 1993.

[6] A. P. Ströle, “Arithmetic Pattern Generators for Built-In Self-Test,” in *Proceedings of the International Conference on CAD (ICCAD)*, pp. 131–134, 1996.

[7] S. Gupta, J. Rajski, and J. Tyszer, “Test Pattern Generation Based on Arithmetic Operations,” in *Proceedings of the International Conference on CAD (ICCAD)*, pp. 117–124, 1994.

[8] S. Gupta, J. Rajski, and J. Tyszer, “Arithmetic Addaptive Generators of Pseudo-Exhaustive Test Patterns,” *IEEE Transactions on Computers*, vol. 8, pp. 939–949, August 1996.

[9] A. P. Ströle, “BIST Pattern Generators using Addition and Subtraction Operations,” *Journal of Electronic Testing Theory and Applications (JETTA)*, vol. 11, pp. 69–80, August 1997.

[10] J. Rajski and J. Tyszer, “Multiplicative Window Generators of Pseudo-Random Test Vectors,” in *Proceedings of the European Design and Test Conference (EDTC)*, pp. 42–48, 1996.

[11] B. Krishnanmurthy, “A Dynamic Programming Approach to the Test Point Insertion Problem,” in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 695–705, ACM/IEEE, June 1987.

[12] N. A. Touba and E. J. McCluskey, “Test Point Insertion Based on Path Tracing,” in *Proceedings of the VLSI Test Symposium (VTS)*, pp. 2–8, IEEE, 1996.

[13] H.-J. Wunderlich and G. Kiefer, “Bit-Flipping BIST,” in *Proceedings of the International Conference on CAD (ICCAD)*, pp. 337–343, ACM/IEEE, November 1996.

[14] G. Kiefer and H.-J. Wunderlich, “Using BIST Control for Pattern Generation,” in *Proceedings of the International Test Conference (ITC)*, pp. 347–355, 1997.

[15] N. A. Touba and E. J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," in *Proceedings of the International Test Conference (ITC)*, pp. 674–682, IEEE, 1995.

[16] N. A. Touba and E. J. McCluskey, "Altering A pseudo-random bit sequence for scan-based BIST," in *Proceedings of the International Test Conference (ITC)*, pp. 167–175, 1996.

[17] M. Chatterjee and D. K. Pradhan, "A Novel Pattern Generator for Near-Perfect Fault-Coverage," in *Proceedings of the VLSI Test Symposium (VTS)*, pp. 417–425, IEEE, 1995.

[18] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," in *Proceedings of the International Test Conference (ITC)*, pp. 120–129, IEEE, IEEE, 1992.

[19] S. Venkataraman, J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," in *Proceedings of the International Conference on CAD (ICCAD)*, pp. 572–577, November 1993.

[20] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern Generation for a Deterministic BIST Scheme," in *Proceedings of the International Conference on CAD (ICCAD)*, pp. 88–94, ACM/IEE, November 1995.

[21] S. Hellebrand, H.-J. Wunderlich, and A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," in *Proceedings of the International Test Conference (ITC)*, pp. 195–204, IEEE, 1996.

[22] B. Koenemann, "LFSR-Coded Test Patterns for Scan Design," in *Proceedings of the European Test Conference (ETC)*, pp. 237–242, 1991.

[23] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pp. 40–45, ACM/IEEE, 1990.

[24] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.

[25] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. C-27, pp. 509–516, June 1978.

[26] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinatorial Benchmark Circuits," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, IEEE, 1985.

[27] F. Brglez, D. Bryan, and K. Kozminski, "Combinatorial Profiles of Sequential Benchmark Circuits," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pp. 1229–1234, IEEE, 1989.

[28] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "Compactest: A Method to generate Compact Test Sets for Combinatorial Circuits," in *Proceedings of the International Test Conference (ITC)*, pp. 194–203, 1991.

[29] G. Tromp, "Minimal Test Sets for Combinatorial Circuits," in *Proceedings of the International Test Conference (ITC)*, pp. 204–209, IEEE, 1991.

[30] H. Huguchi, N. Ishiura, and S. Yajima, "Compaction of Test Sets Based on Symbolic Fault Simulation," in *Synthesis and Simulation Meeting and International Interchange*, pp. 253–262, 1992.

[31] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinatorial Logic Circuits," *IEEE Transactions on Computers*, vol. 14, pp. 1496–1504, December 1995.

[32] F. Somenzi, *CUDD: CU Decision Diagram Package*. Colorado University, ftp://vlsi.colorado.edu/pub/, April 1997. Release 2.1.2.