Bit-Flipping Scan - A Unified Architecture for Fault Tolerance and Offline Test

Michael E. Imhof, Hans-Joachim Wunderlich
Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany, email: {imhof, wu}@informatik.uni-stuttgart.de

***Abstract*—Test is an essential task since the early days of digital circuits. Every produced chip undergoes at least a production test supported by on-chip test infrastructure to reduce test cost. Throughout the technology evolution fault tolerance gained importance and is now necessary in many applications to mitigate soft errors threatening consistent operation. While a variety of effective solutions exists to tackle both areas, test and fault tolerance are often implemented orthogonally, and hence do not exploit the potential synergies of a combined solution.**

The unified architecture presented here facilitates fault tolerance and test by combining a checksum of the sequential state with the ability to flip arbitrary bits. Experimental results confirm a reduced area overhead compared to a orthogonal combination of classical test and fault tolerance schemes. In combination with heuristically generated test sequences the test application time and test data volume are reduced significantly.

***Index Terms*—Bit-Flipping Scan, Fault Tolerance, Test, Compaction, ATPG, Satisfiability**

I. Introduction

The technology evolution of digital circuits is accompanied by two main challenges. To assure product quality offline test is a necessity. Under elevated soft error rates online fault tolerance constantly monitoring operation is of vital importance for reliability [1]. These two challenges require an *efficient hardware test* to cope with manufacturing defects as well as *fault tolerance* to confine transient errors caused by Single Event Upsets (SEUs) altering the sequential state.

Testing a circuit after production and throughout its lifetime to prove the presence of manufacturing defects or wearout effects is one of the most challenging areas in digital circuits. Testing sequential circuits without additional Design for Test (DfT) infrastructure is hard to achieve due to the limited access to the circuit state and the associated high complexity of sequential automatic test pattern generation (ATPG). The most widely adopted DfT infrastructure is scan design [2]. It provides observability and controllability of the circuit state by replacing sequential elements with scannable counterparts and grouping them into scan chains that are read and written sequentially. Nowadays multiple scan chains are used. The ability to use combinational test sets is paid by additional area overhead as well as increased test application times and test data volume. Although solutions like the use of multiple (shorter) scan chains or on-chip test data (de-)compression and compaction [3, 4, 5, 6] are able to reduce the test time and volume, they often substantially increase the area overhead in addition to the overhead introduced by the scan elements.

An alternative to scan-based DfT infrastructure is Random Access Scan (RAS) [7, 8]. It arranges the flip-flops of a circuit in an array providing unique access to read and write single bits. In [9, 10] a toggle flip-flop is used to invert a bit instead of writing it. While still incorporating a high area overhead, the results show that significant savings in test time and volume are possible if the next test pattern is setup by selectively updating the captured circuit response.

Fault tolerance can be achieved by time, space or information redundancy. Due to the non-regular structure of random logic, most schemes protect the sequential state by a combination of time and space redundancy. The RAZOR approach [11] as well as the GRAAL scheme [12] duplicate each bit to detect SEUs and correct them by restoring the value from the shadow element. The area overhead inherent to bitwise duplication and comparison is reduced by using latches. If present, the scan portion can be reused to implement the shadow elements, however this implies that it runs at speed.

The work presented here targets the convergence of test and system reliability solutions by the following contributions:

- A **Unified Architecture** (Fig. 1) utilizing information and structural redundancy. Each register R_i is extended with a checksum computation, a checksum register C_i and a mechanism to flip individual bits. Thereby enabling
  - *Fault Tolerance* by effectively protecting the sequential state against SEUs.
  - *Test Access* by observing compacted register states and controlling arbitrary register bits without the use of full scan.

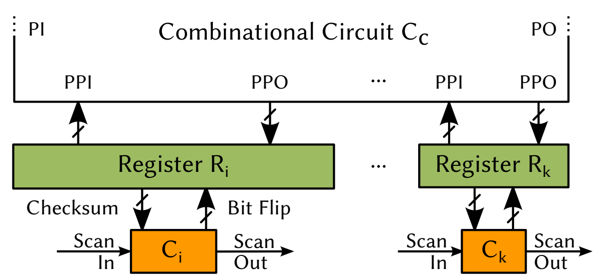

Fig. 1. Presented Unified Architecture

- A heuristic for generating a **Bit-Flipping Scan Test Sequence** consisting of test patterns that validate the compacted sequential circuit state (test response) and setup the sequential state for the next test pattern with a minimized amount of bit-flips.

The next two sections detail the architecture and its use for fault tolerance and test access. Sections IV and V explain how the Bit-Flipping Scan test sequence generation is modeled as a satisfiability problem and solved heuristically.

978-3-9815370-2-4/DATE14/©2014 EDAA

II. Online Fault Tolerance Architecture

The online fault tolerance architecture from [13] is slightly extended to serve as the foundation for an efficient offline test. It protects the sequential state stored in registers against SEUs (Fig. 2). For each register R_i a combination of information and structural redundancy is employed to derive a resident checksum and store it in an additional register C_i. SEUs are detected by a signature S_i computed as the difference between the stored resident checksum and the checksum recomputed from the register values C_i'. Detected SEUs are localized by decoding the signature. Finally, the clock is gated and the affected register bit is corrected in one additional clock cycle with the help of a sequential standard cell that is inherently able to invert its internal state. False corrections due to SEUs in C_i are prevented by a parity of C_i. For offline testing scan design is added to C_i and the decoder is gated by the scan enable signal.

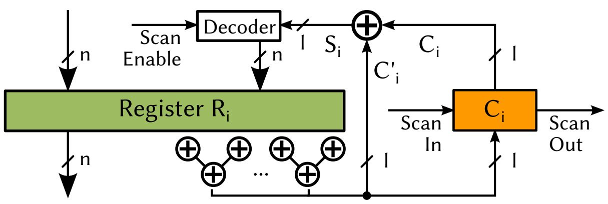

Fig. 2. Fault Tolerance Architecture with Test Extensions

The following subsections discuss the two main underlying concepts used to implement the fault tolerance: A) The area efficient *error detecting and correcting (EDAC) code* computation of the register content, and B) the efficient correction of SEUs at bit level employing *Bit-Flipping Flip-Flops*.

A. Error Detecting and Correcting Code

Let R_i be a register with n bits. Let $R_i = [r_n, \cdots, r_1]^T$ represent the data word vector in matrix notation where r_{adr} ($n \geq adr \geq 1$) references the bit at address adr. The modulo-2 address characteristic proposed in [14] is defined as the bit-wise XOR of all addresses where $r_{adr} = 1$. r_0 is not used, as address 0 does not contribute to C_i.

The mapping between data and characteristic bits corresponds to the generator matrix of a Hamming code and can be expressed by a modulo-2 characteristic matrix M.

$$M = \begin{bmatrix} adr\ n & \cdots & adr\ 1 \end{bmatrix}$$

It consists of l rows and n columns, where n is the number of data bits and each column contains the binary address adr of the associated data bit. The maximum length over all used addresses defines the size of the calculated characteristic and depends on n logarithmically:

$$l = \lceil log_2(n+1) \rceil. \tag{1}$$

The characteristic C_i is computed by multiplying M with R_i:

$$C_i = M \cdot R_i.$$

To detect an error, the characteristic of the original register content R_i is computed at time t_j and stored in an additional register C_i of size l. We call C_i the resident characteristic.

The recomputed characteristic C_i' is then concurrently derived from the register content R_i until new data is written. The difference between the resident characteristic C_i and the recomputed characteristic C_i' is called the signature of R_i:

$$S_i = C_i \oplus C_i'.$$

If S_i is the all-zero vector no deviation was detected, otherwise S_i contains the address localizing the register bit affected by a single bit upset (SBU). The characteristic computation can be efficiently implemented using XOR2 standard cells [15].

Example: Let R_1 be a 7-bit register with value $\left[1011010\right]^T$. Together with the modulo-2 checksum matrix M, the resident characteristic C_1 is computed and stored:

$$C_1 = M \cdot R_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot R_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Now a SBU affects R_1 and flips bit 5, resulting in the faulty register value $R_1' = \left[1001010\right]^T$. The characteristic is recomputed as C_1' and the signature S_1 is calculated:

$$C_1' = M \cdot R_1' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad S_1 = C_1 \oplus C_1' = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

As S_1 is not zero the SBU is detected. Moreover S_1 contains the address 5, thereby correctly localizing the SBU.

During offline test the characteristic is used for test response compaction.

B. Bit Flipping Flip-Flop

Whenever the signature S_i is not the zero vector it is decoded to a n-bit wide correction vector by a 1-out-of-n decoder. The vector then triggers the correction of the erroneous register bit while preserving the state of all other bits.

In contrast to the Bit-Flipping-Latch from [13], the Bit-Flipping Flip-Flop (Fig. 3) targets an edge-triggered design style. The master latch consists of two inverters (INV) and two transmission gates (TG). Both transmission gates are controlled by the control signal pair $\{L, \overline{L}\}$, selecting whether a new value is latched or the internal state is preserved.

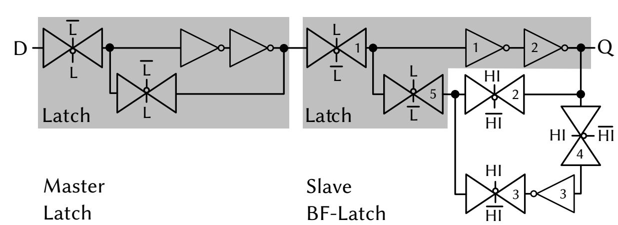

Fig. 3. Bit-Flipping Flip-Flop: Schematic

The slave latch contains an additional inverting feedback loop (TG4, inverter INV3 and TG3) to flip the internal state. The new control signal pair $\{HI, \overline{HI}\}$ for TG2 and TG3 selects either the original or the inverting feedback loop. To avoid metastability of the inverting feedback loop, the inverter INV3 is precharged by TG4 if and only if the loop is not active. Inverting $\{HI, \overline{HI}\}$ while the slave latch stores a value feeds the inverted value of Q to the inverter chain. If the inversion is canceled the non-inverting loop stores the inverted value.

The Bit-Flipping Flip-Flop can be implemented efficiently as a standard cell similar to the bit-flipping latch in [13].

III. Offline Test Access Architecture

The unified architecture is now used for test access to observe and control the sequential circuit state. Therefor the characteristic registers are equipped with scan design and the scan enable signal is used to gate the decoders (Fig. 2). Instead of directly observing the register content R_i in n cycles, the compacted characteristic in register C_i is observed in l cycles. To control R_i the bit-flipping capability inherent to the fault tolerance architecture is reused: Bit-flips are triggered at desired positions by shifting an appropriate characteristic into C_i. The efficiency of the accomplished test access depends on the ratio between n and l as well as the amount of bit-flips.

A. Observing a Test Response

After setting up the stimulus for a test pattern p, the circuit response is captured into the internal registers R. In order to validate if the test pattern passed or failed the test response must be observed. The fault tolerance infrastructure already computes the resident characteristic of each R_i and stores it in the additional register C_i (Fig. 2). Instead of observing the captured response in R_i in n shift cycles, C_i is made scannable and the compacted circuit response is observed in l shift cycles, where $l \ll n$ (Eq. 1).

The value of C_i depends on all bits of register R_i and represents a compacted version of the register content. It has the same properties as a response generated by a dedicated response compactor but reuses existing infrastructure. The characteristic has the same aliasing probability than other SECDED Hamming Codes [14]. It follows, that the compaction quality of Bit-Flipping Scan is comparable to methods like X-Compact [5] or EDT [3].

B. Controlling a Register by Bit-Flipping

A mechanism to flip single bits of a register R_i is present in the architecture to correct SBUs in the fault tolerance mode. Now, this feature is used to setup the next state of register R_i by a series of bit-flips. For each pattern and bit-flip only l bits need to be shifted in, thereby reducing the complexity of the shift operation logarithmically from $O(n)$ to $O(log_2(n))$ (Eq. 1). Fig. 2 shows the involved architecture parts.

Let p_1 and p_2 be two test patterns, let $O(R_i, p_1)$ denote the state of register R_i after the capture cycle of p_1 with both characteristics C_i, C'_i being equal and let $I(R_i, p_2)$ denote the state of R_i needed to setup p_2. Assume without loss of generality, that $I(R_i, p2)$ and $O(R_i, p1)$ differ in exactly one bit at address $adr\ b$ ($1 \leq adr\ b \leq n$), and their Hamming distance is one. Then, the desired register value can be deduced by a single flip of the bit at $adr\ b$.

To trigger a bit-flip at this address the signature S_i needs to encode $adr\ b$: $S_i = adr\ b$. As the register state after p_1 and the associated recomputed characteristic C'_i are known, the resident characteristic C_i is computed as: $C_i = S_i \oplus C'_i$. Scanning in C_i triggers a bit-flip at $adr\ b$, generating $I(R_i, p_2)$ from $O(R_i, p_1)$ with l shift cycles and one additional cycle for the bit-flipping. At the same time, the compacted register state C_i is scanned out and observed. If the Hamming distance between the two register states is larger than one, a series of single bit-flips is used.

C. Efficient Test Access

In traditional scan design the test application time depends on the maximum scan chain length n and the number of patterns. To apply a single pattern p_2 the captured response $O(R_i, p_1)$ of the previous pattern p_1 is scanned out in n cycles while the desired state $I(R_i, p_2)$ for p_2 is shifted in concurrently. Then the circuit state is captured in one additional cycle: $\text{TAT}_S = n + 1$.

For the presented Bit-Flipping Scan scheme, the test time is dominated by the number of bit-flips bf. For each flip, l shift cycles and one flip cycle are needed. After applying all flips the circuit state is captured: $\text{TAT}_{BFS} = bf \cdot (l + 1) + 1$.

Bit-Flipping Scan results in short test times. Formally, the maximum number of bit-flips at which both schemes have the same test time is defined by

$$\text{TAT}_{BFS} \leq \text{TAT}_S \Leftrightarrow bf \cdot (l + 1) + 1 \leq n + 1$$
$$\Leftrightarrow bf \leq \frac{n}{l + 1} \Leftrightarrow bf \leq \frac{n}{\lceil log_2(n + 1)\rceil + 1}$$

Example: For a maximum register size respectively scan chain length of 127 it follows that Bit-Flipping Scan has a lower test time if 15 or less flips per register and pattern are required ($bf \leq 15.875$).

Bit-Flipping Scan facilitates efficient test access by a logarithmic scan chain length reduction and altered scan chain semantics. Without loss of generality, classical test data (de-)compression and compaction schemes can be utilized to further improve test efficiency. The next sections show how the generation of optimized Bit-Flipping Scan test sequences is modeled and solved heuristically.

IV. Modeling the Test Sequence Generation

While in principle any test set can be applied using the test access provided by the unified architecture it is very likely to result in a suboptimal test time and volume due to a high number of involved bit-flips. The goal of efficiently utilizing the unified architecture for offline test is achieved by a tailored sequence of test patterns. After defining the properties of an globally optimal test sequence the reduction of sequential ATPG under bit-flips to a Boolean satisfiability problem and its modeling in conjunctive normal form (CNF) is discussed.

For a circuit C with a set of faults F, an optimal Bit-Flipping Scan test sequence P_{opt} ensures that

- all faults f in the fault universe F are detected by P_{opt}
- the number of bit-flips to setup a register R_i for pattern p_j from the previous register state $O(R_i, p_{j-1})$ is bound by $HammingDist(O(R_i, p_{j-1}), I(R_i, p_j)) \leq bf_{bound}$
- the length of P_{opt} is minimal.

A. Circuit Modeling

A combinational representation C_C of C is built by removing all sequential elements and adding pseudo-primary in- and outputs (PPI/PPO). Each gate $g_i \in C_C$ is then represented in CNF using the Tseitin encoding which generates a linear number of clauses at the cost of introducing a linear number of new variables [16]. Each gate g_i with inputs $i_1, \cdots, i_n$ and output o implementing a Boolean function $o = \phi_g(i_1, \cdots, i_n)$ is logically equivalent to $\Phi_g = (\overline{o} \vee \phi_g(i_1, \cdots, i_n)) \wedge \left(\overline{\phi_g(i_1, \cdots, i_n)} \vee o\right)$. Expanding the equation in a product-of-sums form yields the set of clauses Φ_g in CNF. The circuit C_C is then described in CNF as

$$\Phi_{C_C} = \bigwedge_{g_i \in C_C} \Phi_{g_i} \ .$$

B. Modeling of Stuck-At Faults

Each stuck-at fault in F is represented as a new free literal f. The faulty circuit $\Phi_{c'_f}$ is modeled by copying the output cone c_f of the fault site s_f and assigning new literals to the fault location and all other signals in the fault cone c'_f (s'_f for s_f and $\forall s_n \in c_f : s'_n$). At the edge of the cone the according literals from Φ_{C_C} are used. To generate a test pattern for f with polarity $p_f \in \{0, 1\}$ three conditions need to hold:

- Fault-free circuit: Fault site has the correct value: $s_f = \overline{p_f}$
- Faulty circuit: Fault site has the faulty value: $s'_f = p_f$
- f is observed at least at one output: $obs_f = \bigvee_{\forall (o,o') \in (c_f, c'_f)} (o \oplus o')$.

Then fault f is modeled as

$$\Phi_f = \Phi_{c'_f} \wedge \left(f \vee \overline{(s_f = \overline{p_f})} \vee \overline{(s'_f = p_f)} \vee \overline{(obs_f)}\right) \wedge \\ (\overline{f} \vee (s_f = \overline{p_f})) \wedge (\overline{f} \vee (s'_f = p_f)) \wedge (\overline{f} \vee obs_f) \ .$$

C. Sequential Mapping and Modeling of Bit-Flips

The sequential behavior of C_S is modeled by unrolling. Each timeframe t_j is modeled by Φ_{C_C, t_j} consisting of a copy of Φ_{C_C} with appropriate literal renaming and Φ_{f, t_j} denotes fault f in timeframe t_j.

Bit-Flips are modeled by introducing new free literals $B(R_i, t_j)$ for each register R_i. Together with the pseudo-primary output literals $O(R_i, t_{j-1})$ of the previous timeframe the sequential state in timeframe t_j is modeled:

$$\Phi^B_{t_{j-1}, t_j} = \bigwedge_{\forall R_i \in C_S} (O(R_i, t_{j-1}) \oplus B(R_i, t_j) = I(R_i, t_j)) \ .$$

The sequential behavior under bit-flips is modeled as $\Phi_B = \wedge^x_{\forall j=1}(\Phi^B_{t_{j-1}, t_j})$. The number of bit-flips per register and timeframe is restricted by a cardinality constraint allowing 'atmost' bf_{bound} flip literals per register and timeframe to be true:

$$\Phi^{B_{card}}_{t_{j-1}, t_j} = \bigwedge_{\forall R_i \in C_S} atmost(B(R_i, t_j), bf_{bound}) \ .$$

D. Optimal Test Sequence P_{opt}

The SAT instance for the optimal test sequence P_{opt} from the beginning of this section can now be modeled as follows.

The circuit is unrolled for x timeframes: $\Phi_{C_C} = \Phi_{C_C, t_1} \wedge \cdots \wedge \Phi_{C_C, t_x}$. All faults are added to each timeframe. As it is sufficient to detect a fault once, a disjunction over all timeframes is added per fault: $\Phi_F = (\wedge_{\forall t_j \forall f} \Phi_{f, t_j}) \wedge (\wedge_{\forall f}(\vee_{\forall t_j}(f(t_j))))$. Between consecutive timeframes, bit-flip cardinality constraints are added to limit the maximum number of flips per register: $\Phi_{B_{card}} = \wedge^x_{\forall j=1}(\Phi^{B_{card}}_{t_{j-1}, t_j})$. The literals of timeframe 0 are set to the registers initialization values: Φ_0.

Solving the model $\Phi_{opt} = \Phi_{C_C} \wedge \Phi_F \wedge \Phi_B \wedge \Phi_{B_{card}} \wedge \Phi_0$ yields a solution for x timeframes if it exists. The assignment of literals associated with primary in- and outputs in each timeframe t_i corresponds to a pattern p_i. The generated sequence detects all faults in F with at most bf_{bound} bit-flips per pattern (Sec. III-B). The test sequence P_{opt} with minimum length is found by bisection over the number of timeframes.

Finding the globally optimal test sequence is only feasible for small circuits, small fault universes and a limited number of timeframes due to the high complexity of ATPG and the associated runtimes [17]. Nonetheless an optimized Bit-Flipping Scan test sequence can be generated iteratively as depicted in the next section.

V. Bit-Flipping Scan Test Sequence Generation

The heuristic iteratively generates patterns of the test sequence for combinationally detectable faults from the fault universe F, where each pattern p targets a limited number of not detected faults F_{nd}. All patterns are guaranteed to require a minimal number of bit-flips while covering the maximum amount of faults from F_{nd}. As a preprocessing step, F is classified by combinational ATPG and undetectable faults are removed. The remaining faults are sorted in descending order according to their testability, thereby putting preference on hard to detect faults in the iterative pattern generation.

The heuristic depicted in pseudocode in Algorithm 1 is invoked with the initialization pattern p_0, the fault universe F and the number of concurrently targeted faults $maxF$.

First, the SAT-model Φ (l. 4) is built modeling one timeframe of the combinational circuit (Φ_{C_C}), the limited number of faults contained in F_{nd}, the bit-flips associated to the PPIs Φ_B as well as cardinality constraint $\Phi_{B_{numBF}}$ restricting the number of flips per register to $numBF$.

The for-loop (l. 6-14) searches for a pattern p_j covering a maximum number of faults $numF$ from F_{nd}. In each iteration, for a given number of faults, a cardinality constraint is added to detect at least $numF$ faults: $\Phi_{numF} = atleast(F_{nd}, numF)$. If the model is satisfiable under the current sequential state $\Phi_{p_{j-1}}$, pattern p_j is extracted and the loop continues with an increased $numF$. Once the model is not satisfiable, two cases need to be distinguished:

- A pattern was found in an earlier iteration (l. 15): The current iteration proves that no pattern covering more faults from F_{nd} exists. The list of currently modeled faults F_{nd} and the global fault list F are pruned by fault simulation of p, p_j is added to the pattern sequence P and the bit-flip constraint Φ_{numBF} is reset (l. 16-18). If more

Algorithm 1 Iterative Bit-Flipping Scan ATPG

1: **function** GENERATEBFSPATTERNS($p_0, F, maxF$)
2: $F_{nd} \leftarrow getND(F, maxF)$ ▷ $maxF$ not det. faults
3: $P \leftarrow \emptyset \cup p_0;\ j \leftarrow 1;\ numBF \leftarrow 1$ ▷ init, 1 BF per Reg.
4: $\Phi \leftarrow \Phi_{C_C} \wedge \Phi_{F_{nd}} \wedge \Phi_B \wedge \Phi_{B_{numBF}}$ ▷ model
5: **while** $F \neq \emptyset$ **do**
6: **for** $numF \leftarrow 1, maxF$ **do** ▷ cover max. faults
7: $update(\Phi, \Phi_{numF})$ ▷ add cardinality constraint
8: $SAT \leftarrow solve(\Phi, \Phi_{p_{j-1}})$ ▷ under seq. state of p_{j-1}
9: **if** SAT **then**
10: $p_j \leftarrow extractPattern(\Phi);\ pFound \leftarrow true$
11: **else**
12: break for-loop ▷ (line 6)
13: **end if**
14: **end for**
15: **if** pFound **then**
16: $F \leftarrow fsim(p_j, F);\ F_{nd} \leftarrow fsim(p_j, F_{nd})$ ▷ prune
17: $P \leftarrow P \cup p_j;\ j \leftarrow j+1;\ pFound \leftarrow false$ ▷ add p
18: $numBF \leftarrow 1;\ update(\Phi, \Phi_{numBF})$ ▷ reset
19: **if** $|F_{nd}| < 0.9 \cdot maxF$ **then** ▷ update faults
20: $F_{nd} \leftarrow getND(F, maxF);\ update(\Phi, \Phi_{F_{nd}})$
21: **end if**
22: **else** ▷ no p under $numBF$
23: $numBF \leftarrow numBF + 1;\ update(\Phi, \Phi_{B_{numBF}})$
24: **end if**
25: **end while**
26: **return** P
27: **end function**

than 10% of the faults contained in F_{nd} are detected, the model is rebuilt with the next $maxF$ faults (l. 20).

- No pattern was found at all (l. 22): The iteration proves that no pattern detecting even a single fault exists under the constrained amount of bit-flips. $numBF$ is increased and the for-loop is re-executed, thereby ensuring that the next pattern detects the maximum amount of modeled faults under the increased number of bit-flips.

The surrounding while loop (l. 5-25) terminates when all faults from F are detected. As F contains only combinationally detectable faults each fault in F will be detected, in the last resort by a pattern requiring a high amount of bit-flips.

VI. Experimental Evaluation

The presented scheme is evaluated for ISCAS89 and ITC99 benchmarks as well as for industrial circuits kindly provided by NXP (formerly Philips). For each circuit, the combinational core is synthesized for the 45 nm Nangate Open Cell Library (OCL) [18] using one- and two-input gates.

Four different scenarios are analyzed:

a) Original: D-Flip-Flops (DFF, $4.522\,\mu m^2$).
b) Scan Design: Scannable D-FFs (SDFFR, $6.916\,\mu m^2$).
c) Fault Tolerance + Scan Design (FTScan): Scannable D-Flip-Flops together with a fault tolerance scheme comparable to RAZOR [11] or GRAAL [12]: A shadow latch (DLH, $2.926\,\mu m^2$), an exclusive OR (XOR2, $1.596\,\mu m^2$) and a multiplexer (MUX2, $1.862\,\mu m^2$).
d) Bit-Flipping Scan (BFScan): Bit-Flipping Flip-Flops implemented as a new OCL-compatible standard cell (BFF, $5.054\,\mu m^2$) combined with the characteristic computation and the signature decoder as well as scannable characteristic registers (SDFFR, $6.916\,\mu m^2$).

For b) and c), all FFs are organized into scan chains with a maximum length of 127. For d), a register is implemented for each chain from the scan chain configuration used in b) and c). Note, that a chain length of 127 is rather short. For longer chains, scenarios b) and c) will scale linear in terms of area and test time. The area of the unified architecture and the test time of Bit-Flipping Scan sequences will grow slower due to the logarithmic correlation between n and l (Eq. 1).

A. Area Overhead

The gate area of the synthesized original circuit in μm^2 is used as a baseline in Table I, corresponding gate counts can be found in columns 2 & 3 of Table II. The implementation of scan design increases the area (col. 3) and the area overhead to the original circuit is between 4.3% and 24.7% (col. 4). Implementing fault tolerance by bitwise redundancy orthogonal to scan design further increases the area (cols. 5 & 6). The area associated with the presented unified architecture (col. 7) is moderate for all circuits with an overhead between 21.7% and 81.8% (col. 8). The last column depicts the difference between the overheads of BFScan (col. 8) and FTScan (col. 6).

The results show, that compared to an orthogonal combination of the two classical methods, the unified architecture targeting both test and fault tolerance uses less area.

B. Test Application Time

Table II compares the test application times. A highly compacted test set is generated for each FTScan configuration using a commercial ATPG. The heuristic from Section V is used to generate BFScan test sequences, where $maxF$ was set to 100, providing a good tradeoff between runtime and achieved test time.

The total number of clock cycles for Bit-Flipping Scan is significantly lower compared to FTScan (col. 12 & 7) although the BFScan test sequence contains more patterns (cols. 8 & 4). Instead of n shift cycles only $\lceil log_2(n+1) \rceil$ cycles are used per pattern and bit-flip, allowing to apply more patterns in

TABLE I
AREA OVERHEAD FOR A MAX. CHAIN LENGTH/REGISTER SIZE OF 127

	Original	Scan Design		FT + Scan		Bit-Flipping Scan		**Area**
Circuit	(DFF)	(SDFFR)	OH	(SDFFR)	OH	(BFFF)	OH	**ΔOH**
name	μm^2	μm^2	+%	μm^2	+%	μm^2	+%	**+%**
(1)	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	*(7)*	*(8)*	***(8)-(6)***
s35932	13461	16781	24.7	28946	115.0	24469	81.8	**-33.3**
s38417	14444	17883	23.8	29364	103.3	24344	68.5	**-34.8**
s38584	15744	18433	17.1	28582	81.5	24410	55.0	**-26.5**
b14	5252	5553	5.7	7390	40.7	6531	24.3	**-16.3**
b17	27606	30140	9.2	40096	45.2	36276	31.4	**-13.8**
b20	11048	11541	4.5	15016	35.9	13469	21.9	**-14.0**
b22	16531	17241	4.3	22740	37.6	20117	21.7	**-15.9**
p35k	27566	31971	16.0	45927	66.6	39649	43.8	**-22.8**
p45k	29346	33759	15.0	50125	70.8	43465	48.1	**-22.7**
p78k	57561	64935	12.8	83116	44.4	73647	27.9	**-16.5**
p100k	71177	82250	15.6	121278	70.4	104517	46.8	**-23.5**
p141k	128672	148146	15.1	220212	71.1	189140	47.0	**-24.1**
p239k	209954	243963	16.2	373044	77.7	317872	51.4	**-26.3**
p259k	246661	279568	13.3	419886	70.2	361851	46.7	**-23.5**
p267k	177598	210587	18.6	325096	83.1	276449	55.7	**-27.4**
p269k	177753	210564	18.5	325139	82.9	276912	55.8	**-27.1**
p279k	207302	240778	16.1	363884	75.5	311310	50.2	**-25.4**
p295k	211510	245279	16.0	372464	76.1	322304	52.4	**-23.7**
p330k	210958	244121	15.7	357882	69.6	308389	46.2	**-23.5**
p378k	287773	324514	12.8	413825	43.8	365445	27.0	**-16.8**
p418k	317313	371812	17.2	570179	79.7	489276	54.2	**-25.5**

TABLE II
TEST APPLICATION TIME (TAT) AND TEST DATA VOLUME (TDV) FOR A MAXIMUM CHAIN LENGTH/REGISTER SIZE OF 127

Circuit			Test Application Time										Test Data Volume		
	Gates		Fault Tolerance + Scan Design				Bit-Flipping Scan					**TAT**			**TDV**
name *(1)*	Comb. *count* *(2)*	Seq. *count* *(3)*	Pat. *count* *(4)*	Capture *cycles* *(5)*	Scan *cycles* *(6)*	Sum *cycles* *(7)*	Pat. *count* *(8)*	Capture *cycles* *(9)*	Flip *cycles* *(10)*	Scan *cycles* *(11)*	Sum *cycles* *(12)*	**Speedup** ***ratio*** ***(13)***	FTScan *bits* *(14)*	BFScan *bits* *(15)*	**Reduction** ***%*** ***(16)***
s35932	16065	1728	50	50	6477	6527	115	115	114	805	1034	**6.31**	190550	63365	**66.75**
s38417	23861	1636	150	150	19177	19327	711	711	716	5013	6440	**3.00**	510900	224531	**56.05**
s38584	21938	1452	173	173	22098	22271	618	618	626	4389	5633	**3.95**	550672	281761	**48.83**
b14	10681	245	565	565	71882	72447	1293	1293	1948	12300	15541	**4.66**	325440	148259	**54.44**
b17	35482	1415	661	661	84074	84735	2735	2735	3952	27437	34124	**2.48**	1959204	812364	**58.54**
b20	21599	490	568	568	72263	72831	2043	2043	2756	18730	23529	**3.10**	587312	229981	**60.84**
b22	32090	735	528	528	67183	67711	2095	2095	2558	17732	22385	**3.02**	804672	294639	**63.38**
p35k	41443	2173	1068	1068	135763	136831	2726	2726	2883	20165	25774	**5.31**	5490588	2838510	**48.30**
p45k	38811	2331	2100	2100	266827	268927	3069	3069	3216	22504	28789	**9.34**	13206900	5799212	**56.09**
p78k	68263	2977	94	94	12065	12159	124	124	123	868	1115	**10.90**	623408	125488	**79.87**
p100k	84356	5735	2050	2050	260477	262527	4356	4356	5483	38369	48208	**5.45**	24048550	3816114	**84.13**
p141k	152808	10501	612	612	77851	78463	1620	1620	1621	11354	14595	**5.38**	13336704	3162100	**76.29**
p239k	224597	18382	517	517	65786	66303	1663	1663	1668	11683	15014	**4.42**	19225490	4070658	**78.83**
p259k	298796	18398	672	672	85471	86143	2218	2218	2218	15533	19969	**4.31**	25003776	5415952	**78.34**
p267k	238697	16528	724	724	92075	92799	2834	2834	2841	19901	25576	**3.63**	24581972	7726117	**68.57**
p269k	239771	16528	726	726	92329	93055	2891	2891	2896	20279	26066	**3.57**	24650604	7883099	**68.02**
p279k	257736	17524	780	780	99187	99967	2912	2912	2911	20384	26207	**3.81**	28002402	8109920	**71.04**
p295k	249747	18465	1579	1579	200660	202239	6615	6615	6614	46305	59534	**3.40**	58468791	14162715	**75.78**
p330k	312666	16775	1752	1752	222631	224383	5071	5071	5075	35532	45678	**4.91**	62157456	19187520	**69.13**
p378k	341315	14885	87	87	11176	11263	234	234	233	1638	2105	**5.35**	2884224	1177020	**59.19**
p418k	382633	28616	875	875	111252	112127	3526	3526	3527	24696	31749	**3.53**	52707802	21747326	**58.74**

shorter time. Thus a raised coverage of non-target faults could be achieved, but needs to be investigated further.

The results show, that Bit-Flipping Scan results in a test application time speedup of 2.48X in the worst and up to 10.9X in the best case (col. 13).

C. Test Data Volume

The reported test data volume includes all bits exchanged with the circuit over primary and pseudo-primary in- and outputs. The test volume of Bit-Flipping Scan is lower for all circuits (col. 15 & 14). Column 16 depicts the achieved test volume reduction. For s35932, the test volume of BFScan is just 33.25% of the original volume. Thus Bit-Flipping Scan reduced the original volume by 66.75%.

The results show, that Bit-Flipping Scan reduces the test volume by between 48.3% and 84.13% of the original test volume (col. 16).

VII. CONCLUSION

A unified architecture was presented that can be used for fault tolerance and offline test. It combines a checksum of the sequential circuit state with the ability to flip arbitrary bits. In fault tolerance, Single Event Upsets affecting the sequential elements are detected and located. A correction is performed in one additional clock cycle. In test, compacted test responses are observed and bit-flipping is used to derive the next test pattern from the captured state. The experimental results confirm a reduced area overhead due to the integrated consideration of fault tolerance and test. The presented test sequence generation heuristic successfully exploits the architectures capabilities and results in a significant reduction of test application time and test data volume.

VIII. ACKNOWLEDGMENT

Parts of this work were supported by the German Research Foundation (DFG) under grants WU 245/13-1 (RM-BIST). We would like to thank Manuel Jerger for his support.

REFERENCES

[1] *International Technology Roadmap for Semiconductors (ITRS)*, 2012.

[2] M. J. Y. Williams and J. B. Angell, "Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic," *IEEE Transactions on Computers*, vol. 22, pp. 46–60, 1973.

[3] J. Rajski, J. Tyszer, M. Kassab, *et al.*, "Embedded deterministic test," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 23, no. 5, pp. 776–792, 2004.

[4] N. A. Touba, "Survey of test vector compression techniques," *IEEE Design & Test of Computers*, vol. 23, no. 4, pp. 294–303, 2006.

[5] S. Mitra and K. S. Kim, "X-compact: an efficient response compaction technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 3, pp. 421–432, 2004.

[6] M. A. Kochte, S. Holst, M. Elm, *et al.*, "Test Encoding for Extreme Response Compaction," in *IEEE Europ. Test Symp. (ETS)*, 2009, pp. 155–160.

[7] H. Ando, "Testing VLSI with random access scan," *Proceedings of Compcon 80*, pp. 50–52, 1980.

[8] D. H. Baik, K. K. Saluja, and S. Kajihara, "Random Access Scan: A solution to test power, test data volume and test time," in *17th International Conference on VLSI Design (VLSID)*, 2004, pp. 20–00.

[9] A. S. Mudlapur, V. Agrawal, and A. D. Singh, "A random access scan architecture to reduce hardware overhead," in *IEEE International Test Conference (ITC)*, 2005, pp. 1–9.

[10] R. Adiga, G. Arpit, V. Singh, *et al.*, "Modified T-Flip-Flop based scan cell for RAS," in *IEEE Europ. Test Symp. (ETS)*, 2010, pp. 113–118.

[11] D. Ernst, N. Kim, S. Das, *et al.*, "Razor: a low-power pipeline based on circuit-level timing speculation," *IEEE/ACM International Symposium on Microarchitecture*, pp. 7–18, 2003.

[12] M. Nicolaidis, "GRAAL: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies," *IEEE International Test Conference (ITC)*, pp. 1–10, 2007.

[13] M. E. Imhof and H.-J. Wunderlich, "Soft Error Correction in Embedded Storage Elements," in *17th IEEE International On-Line Testing Symposium (IOLTS)*, 2011, pp. 169–174.

[14] V. N. Yarmolik, S. Hellebrand, and H.-J. Wunderlich, "Self-adjusting output data compression: An efficient BIST technique for RAMs," in *Design, Automation and Test in Europe (DATE)*, 1998, pp. 173–179.

[15] M. E. Imhof, H.-J. Wunderlich, and C. G. Zoellin, "Integrating Scan Design and Soft Error Correction in Low-Power Applications," in *14th IEEE Intl. On-Line Testing Symposium (IOLTS)*, 2008, pp. 59–64.

[16] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Studies in constructive mathematics and mathematical logic*, vol. 2, no. 115-125, pp. 10–13, 1968.

[17] O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," *IEEE Trans. on Comp.*, vol. 100, no. 3, pp. 242–249, 1975.

[18] "Nangate Open Cell Library v1.3," http://www.nangate.com.