ATPG Method with a Hybrid Compaction Technique for Combinational Digital Systems

Abdul Rafay Khatri, Ali Hayek, and Josef Börcsök
Department of Computer Architecture and System Programming
University of Kassel, Kassel, Germany
Email: arkhatri@student.uni-kassel.de

***Abstract*—In this paper, the Test Pattern Generation (TPG) with a new simple hybrid (dynamic and static) compaction technique for combinational logic circuits and systems is presented. Digital systems are implemented nowadays on an advance VLSI technology, which is called Field Programmable Gate Array (FPGA). The test procedure requires a deliberate introduction of faults in the System Under Test (SUT). FPGA circuits and systems are developed and written in Hardware Description Languages (HDLs). In addition, a novel method is also proposed and implemented in order to intentionally inject faults in the Verilog HDL code design of the SUT. It covers all possible fault locations in the SUT. This approach provides high quality compact test vectors and efficient memory utilization in comparison with other state-of-the-art methods. Not only are the hardness of individual faults and the sensitiveness of fault locations determined, but also their effectiveness is proposed in the fault tolerance strategy.**

***Keywords*—test pattern generation; dynamic compaction; static compaction; fault coverage; hardness**

I. Introduction

The testing of Very Large Scale Integrated (VLSI) digital systems is a major challenge nowadays. With the increasing complexity of components, it is difficult to guarantee an acceptable degree of reliability and an improving yield of products. Due to the complexity of VLSI circuits, the cost of testing for such systems increases rapidly [1]. Several methods have been proposed to test digital systems and are commonly known as test pattern generation methods. The purpose of these test methods is to ensure that the system is free from defects or faults, meet claimed specifications, and improve yields. The main purposes of the TPG are finding the minimum number of test vectors (i.e. reducing the memory), performing the test operation in a short time (i.e. increasing the speed) and being cost effective [2]. Many TPG methods have been developed so far and can be categorized into exhaustive, pseudo random, algorithmic (D-algorithm [3] PODEM [4] [5], FAN [6], TOPS [4], SOCRATES [7][10]), and simulation based. Besides, the Boolean Satisfiability (SAT) approach is also used as an Automatic TPG (ATPG) method [8]–[11]. Some methods also exist in which test vectors are found by simulation and, furthermore, emulation approaches do exist using the FPGA. The features that FPGA offer are reconfigurability, higher speed, better performance and cost effectiveness [16]. The development of a SUT on the FPGA passes through a number of stages. Usually, a circuit is designed and the code is written in the hardware description languages i.e. Verilog and VHDL, which represent the higher level of abstraction in the development process. Other steps include synthesis, translate, map, place and route, and bit file generation. The deliberate injection of faults is a key step in the test procedure and dependability analysis of a system. In this work, fault injection technique is chosen at the higher level of abstraction, i.e. in the coding phase. For that purpose, a program is written in MATLAB, which accepts a Verilog design file of the SUT, and injects faults at every possible location, as described in Section III. The performance criteria for a TPG are Fault Coverage (FC), fault efficiency, test vector length, and the CPU time [4]. To date, different simulation based TPG methods have been developed and implemented, for example, serial, parallel, deductive and concurrent fault simulation methods, to name a few. The number of test vectors has to be further reduced in order to save memory for the Automated Test Equipment (ATE).

Two techniques are generally used for this purpose, compression and compaction [12]. In this work, the compaction technique, which is further divided into two techniques i.e. dynamic and static, have been focused. The dynamic compaction is a technique in which compact test vectors are generated during the TPG process. The dynamic compaction is a part of the Automatic Test Pattern Generation (ATPG) tools [13]–[20]. In a static compaction technique, the number of test vectors is reduced after their generation. The static compaction technique does not alter or modify the process of TPG, and it can be developed in any programming language or tool. Therefore, the static compaction is cheaper than the dynamic compaction [21], [22].

This paper presents a new TPG method with a simple hybrid compaction technique. The instrumentation based technique in which an extra circuit is added and it is used to inject faults in the target system [23], [24]. When activated, this additional circuit injects faults in the system, whereas it does nothing in the deactivated state. In the proposed approach, one golden (fault-free) model of the SUT, written in Verilog HDL, and other three or more faulty copies of the SUT, in which faults are distributed evenly, are considered as described in Section IV. The flexibility to use an arbitrary fault model has been provided. This method is tested for the stuck-at (1 & 0) fault models separately and the bit flip fault model [24], [25]. Moreover, a new approach to fault injection, selection and activation (FISA) unit is proposed in this paper. The FISA unit is also instrumented in faulty copies of the SUT using MATLAB, which automatically modifies the Verilog code by injecting and distributing faults evenly in all three or more copies of the system. This unit reduces the number of fault selection and activation inputs. The simple and easy-to-understand hybrid compaction (dynamic and static) algorithms are also described in this paper. Our proposed method results

in speeding up the fault detection process, and it determines the most compact set of test vectors of different fault models for their maximum fault coverage.

The organization of this paper is as follows: The related work is presented in Section II. Section III describes the proposed automatic fault injection method in the target system. Section IV describes the proposed approach to the test pattern generation and hybrid compaction method. Section V illustrates experimentation, implementation and discussion about the results. Finally, Section VI concludes the paper and presents the directions for future work.

II. Related Work

Automatic test (commonly known as ATPG) methods have been developed for generating test vectors. For a SUT, the test pattern generation process must be fast, and high quality test vectors should be achieved [5]. The fault simulation serves for different purposes, e.g. generating test data, evaluating test set and circuit testability, providing information for testers, finding faults in a circuit, diagnosing fault locations, and many other applications [5]. In our approach, faults are injected in the Verilog HDL code and the SUT is simulated in $XilinxISIM$.

In the last few decades, testing has been a huge problem for test and design engineers. A plethora of approaches have been presented in the literature but still there is a space to work in this field to make test generation more flexible, reliable, fast, memory efficient and simple to understand. The authors in [26] describe the method of generating minimized test patterns for logic circuits. According to them, the fault simulation is carried out in such a way that when a fault is selected and the input patterns are applied, they should be able to find or detect at least one fault to qualify as a test vector.

One of the approaches used as an ATPG is the Boolean Satisfiability, commonly referred to as the SAT approach [9]–[11], [27]. The SAT solver works on the Boolean formula in the Conjunctive Normal Form (CNF), and the given circuit is transformed into CNF. This approach has few flaws as well, such as the overhead of CNF transformation, missing support of multiple-valued logics and over-specified solutions [27]. Test patterns are also generated using meta-heuristic algorithms, such as genetic algorithm, which are much better than deterministic algorithms [28]–[30]. In some literature, a test cube-aware dynamic compaction scheme is presented for reducing the number of test vectors [18]. In this approach, the generated cubes are merged locally and then switched to compacted cube merging.

A similar approach to the presented work, in which faults are injected using multiplexer logic, is introduced in [31]. In this method, when thousands of faults are injected, the same number of fault select signals are required accordingly, which is certainly not a feasible idea. Furthermore, faults are injected using a multiplexer logic, whereas in our approach, two-inputs AND, OR, and XOR gates are used for stuck at 0,1 and bit flip fault models, respectively. Therefore, the area overhead seems to be reduced.

III. Fault Injection Methodology

Different types of the faults exist, namely permanent, transient, and intermittent faults. A fault is basically an imperfection in a hardware or a software functionality. Some fault models are widely used in the fault injection process, e.g. stuck-at model [4] and bit flip model [25]. These models serves to pretend that an actual fault is occurring in the system because of heavy ion radiations or some other system malfunctions. In order to generate test vectors for a particular system, we need to deliberately inject faults in the system. These faults can be injected anywhere in the synthesizable design netlist, bit file or in the code. However, the faults are injected in the Verilog code of the SUT in this approach. For that purpose, the automatic fault injection method is designed and described in the sequel.

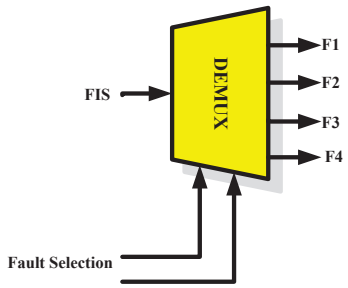

Fig. 1: Proposed DEMUX fault injection model (FISA unit)

A. Automatic Fault Injection and HDL Code Modification

In order to generate test vectors, different faults need to be deliberately injected into the system and later detected. As a fault can occur anywhere in the system, all possibilities of fault occurrence have to be covered for getting compact vectors with maximum fault coverage. It is very cumbersome to manually place individual faults in each line of the Verilog code of the SUT. For that reason, a tool has been developed in MATLAB, which modifies the HDL code and injects faults at all possible locations of the SUT. According to the code modification methodology, this tool also generates a different number of copies of the SUT with equal distribution of faults within them. Additionally, it adds an FISA unit to each copy of the SUT. The FISA unit is based on the DEMUX technique as shown in the Fig.1. When the fault selection input is applied with all possible combinations, faults are sequentially activated, selected and injected one at a time in the SUT. The Fault Selection (FS) input lines can be calculated according to the Eq (1),

$$FS = \lceil \log_2(N) \rceil \tag{1}$$

where FS and N are the fault selection input lines and the number of faults injected per copy of the SUT, respectively.

The developed MATLAB tool accepts a Verilog file (*.v) as an input and analyse it. It calculates all possible locations and divides them into three or more copies as required or permitted on the FPGA. These operators (^ = bit flip, | = stuck at 1, & = stuck at 0) are specified as fault models in the Verilog code. The number of copies of the SUT is limited to a minimum of three. The original and modified code for `C-17.v` from the ISCAS'85 benchmark circuit is shown in Fig.2.

The copy 1 with four bit flip faults is shown in the code. The copy 2 contains bit flip faults in the next two lines, whereas the first two lines remain the same as in the original code

```
// Verilog (Golden copy)
// c17
// Original code
module c17 (N1,N2,N3,N6,N7,N22,N23);
input N1,N2,N3,N6,N7;
output N22, N23;
wire N10,N11,N16,N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule

module c17_1 (select,N1,N2,N3,N6,N7,
N22_f1,N23_f1);
input N1,N2,N3,N6,N7;
input[1:0] select;
output N22_f1, N23_f1;
wire fis=1;
reg f0,f1,f2,f3;
// FISA unit
always @ (select) begin
if (select == 2'd0) begin
f0=fis;f1=0;f2=0;f3=0; end
else if (select == 2'd1) begin
f0=0;f1=fis;f2=0;f3=0; end
else if (select == 2'd2) begin
f0=0;f1=0;f2=fis;f3=0; end
else if (select == 2'd3) begin
f0=0;f1=0;f2=0;f3=fis; end
else begin
f0=0;f1=0;f2=0;f3=0; end
end
wire N10,N11,N16,N19;
// faults in copy 1.
nand NAND2_1 (N10,f0 ^ N1,f1 ^ N3);
nand NAND2_2 (N11,f2 ^ N3,f3 ^ N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22_f1, N10, N16);
nand NAND2_6 (N23_f1, N16, N19);
endmodule
```

Fig. 2: The original and modified code

and so on. According to the Eq(1), FS is calculated and the Select variable is assigned a vector length in the code as [FS-1:0] select in each copy. The modified and instrumented code along with FISA unit is shown in red color.

IV. Test Pattern Generation Methodology and Hybrid Compaction Scheme

In this paper, two main issues have been focussed, i.e. time (or speed) and memory reduction (achieved by obtaining a minimum number of test vectors). This method can be used for any fault models in the target system. A SUT with faults can be simulated and emulated on the FPGA. In order to make the system faster, a minimum of three faults are taken into consideration at a time. Our methodology provides one golden copy (fault free) of the SUT and three or more copies of the golden copy, in which faults are inserted and distributed equally in all copies during the design phase in the code, as described in Section III. In this paper, the SA-(0,1) and the bit flip fault models are considered individually because one of the prime objectives is to find the sensitive areas in the design for aforementioned fault models along with the test vectors. The qualified test vectors and their respective detected faults are stored in this approach.

This proposed method is simple to understand, easy to implement and it is applicable to both simulation and emulation of the SUT.

A. Test Pattern Generation

The block diagram of the proposed TPG approach is shown in Fig.3. The essential blocks are described in detail below.

1) Input Pattern Block: This block is used to generate different input patterns which are simultaneously applied to both the original and faulty copies of the SUT. If the SUT has 6 or less inputs, an exhaustive test approach is applied. However, if the number of inputs is higher than 6, random test approach is applied. Table I shows different SUTs with the corresponding number of the logic gates, number of I/Os and the applied test methodology. The memory is allocated to store 100 qualified test vectors only. These 100 vectors are obtained using dynamic compaction which will be discussed later in detail.

When an input pattern is applied to the target circuit, i.e. golden and faulty copies, both at a time, the select input of the FISA unit will inject and activate each fault one by one for this pattern. In this way all faults are injected, activated and their behaviours are monitored in relation to the applied input.

TABLE I: Characteristics and test methods applied to various SUT

SUT	No. of Logic gates	No. of Inputs	No. of Outputs	Test Methods
C17	6	5	2	Exhaustive
C432	160	36	7	Random
C499	202	41	32	Random
C880	383	60	26	Random
C1355	–	–	–	Random
C1908	880	33	25	Random

2) Golden and Faulty Circuits: In order to detect a fault or to monitor its behaviour, the responses of golden and faulty circuits must be compared before and after an activation of the fault. The FISA unit is also injected automatically in the Verilog code for the faulty circuits, which activates faults by $select$ input.

3) Comparator Block: This block compares the responses of the golden circuit with the response of each faulty copy. The number of output pins of the comparator blocks depends on the number of faulty copies. We have selected more numbers of copies for moderate circuits than large circuits. The minimum numbers of copies are three. Different number of copies would change the fault select lines and speed up the process. When

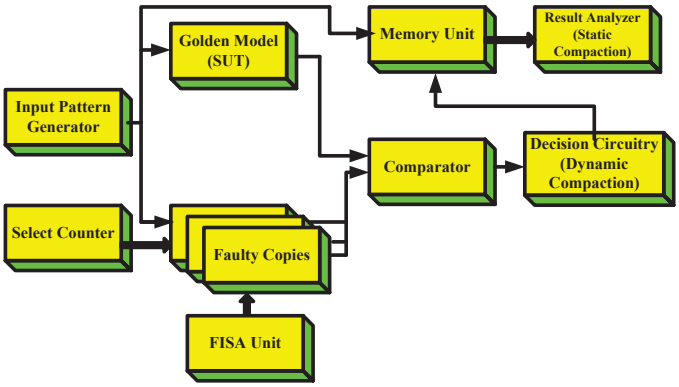

Fig. 3: Block diagram of the proposed approach

the output of comparator is logic '1', it means that the activated fault is detected, and when the comparator output is logic '0', it means the fault is not detected.

B. Decision Circuitry (Dynamic Compaction)

As described earlier, the dynamic compaction is part of an ATPG process. In Fig.3, the block named decision circuitry performs dynamic compaction. The dynamic compaction technique is very simple and easy to implement. A set point value is placed for the qualification of input patterns to test vectors and stored them in the memory. The concept of the set point value is that only those test vectors, which can detect more faults are considered so that a reduced number of test vectors is achieved for a higher FC. The threshold for the set point value is considered to range from 30% to 50% of total faults for various fault models.

For three copies of the SUT, three outputs of the comparator block are stored in a fault array. The size of the fault array is equal to the total number of faults injected. In dynamic compaction, the number of 1's in the fault array are counted and the sum is compared with the set point value. If the sum is equal to or greater than the set point value, the input pattern will be stored as a qualified test vector in the memory, which will be further processed in the stage of static compaction.

C. Memory Block

The memory block consists of rows and columns. The number of rows depends on the pattern applied (in our case 100 for the circuit having 7 or greater inputs), and the number of columns depends on the total number of faults injected, and the number of inputs of the SUT. In this block, the qualified input test patterns from the dynamic compaction step are stored along with the array of faults. The resulting data are transmitted to the result analyzer for the static compaction.

D. Result Analyzer (Static Compaction)

The result analyzer receives data from the memory block and does the necessary operations in order to find the minimum number of test vectors. This block is implemented on the host PC. A new algorithm is proposed for the result analyzer based on the static compaction and is written in MATLAB. The algorithm works as follows.

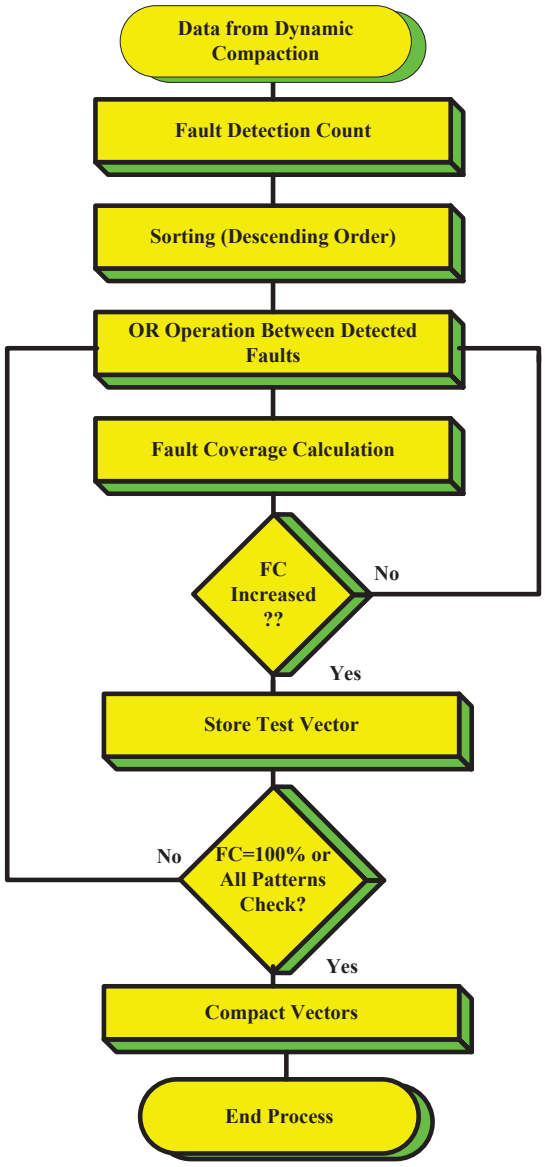

Fig. 4: Flow chart of the static compaction approach

1) The memory unit sends data to the host computer, which is obtained by dynamic compaction.
2) The detection of faults from the fault array is counted for each test vector.
3) The vectors are sorted in the descending order according to the number of faults found by a test vector.
4) Having sorted the vectors, number of faults, and arrays in the descending order, the first memory location is ORed with the rest of the memory locations one by one.
5) The FC is calculated each time. If the FC increases from the previous value, the test vector should be considered for the compact test input. Otherwise, this test vector is removed from the list of dynamically compacted test vectors.
6) Step 4 is repeated till the last memory location has been tested or the FC reaches 100%.
7) The compact test vectors are obtained after static

compaction. The results for various circuits are described in Section V.

The flow chart for the algorithm is shown in Fig.4. This results in the compact test vectors for the maximum fault coverage. The FC can be calculated as

$$FC = \frac{\text{Number of faults detected}}{\text{Number of faults injected}} \times 100\% \qquad (2)$$

V. Experimentation and Implementation

This approach has been verified on the combinational logic circuits from ISCAS'85 test benchmark circuits, whereas the work on synchronous sequential circuits is in progress. These benchmark circuits are available in different HDL languages like Verilog and VHDL. Verilog HDL is considered in this paper. The operation and analysis of the circuit $C-17$ is explained here. The developed automatic fault injection tool in MATLAB accepts the file $C-17.v$ and inserts faults in all possible fault locations. Additionally, the FISA unit is also instrumented as described above. We consider a minimum of three copies for this benchmark circuit. The MATLAB tool is capable of generating any number of copies with evenly distributed faults in them. $XilinxISE13.4$ is used for developing the project. The golden and the faulty copies are combined under the file name $top.v$. This $top.v$ file contains the interconnection of the SUT and all faulty copies, logic for the comparator block, the dynamic compaction technique, and the memory unit. For the simulation, a test bench is written to generate input patterns. The Linear Feedback Shift Register (LFSR) based input generator is designed in the test bench. When an input pattern is first applied and the select input through counter is incremented by one, results are compared for each increment. The dynamic compaction is then applied and the results are stored in memory. For random testing, when the inputs of the SUT are equal to or greater than 7, then 10 different seeds are selected and 10 vectors after dynamic compaction are stored for each seed. The memory unit can store 100 vectors for random test patterns with their fault array. These vectors are further compacted using the static compaction.

The benchmark circuit $C-17$ has five inputs, hence an exhaustive test method is applied. The $C-17$ circuit contains 12 fault locations, 4 per copy, therefore, only two fault select lines are required. When first input pattern 00000 is applied and select input is 00, the first fault is injected and activated in each copy. The outputs are compared and stored in the fault array in the first three locations. Afterwards, the select input is incremented from '00' to '01'. New faults are inserted and activated, outputs are compared and stored in the next three locations in the fault array. The process continues until all faults are injected for the input pattern '00000'. After the fault array is filled completely, the number of 1's are counted and compared with the set point value. For the bit flip fault model experiment, the set point value is 5 for 12 faults. Only 24 vectors are stored in memory out of 32 applied patterns. These vectors detect more faults than the other patterns. These vectors are then used for the static compaction process, which not only does save memory, but also the computation time. The set point value will be different for each fault model because the distribution of fault detection is different for each fault model. In Fig.5 the FC is shown for three fault models. The original

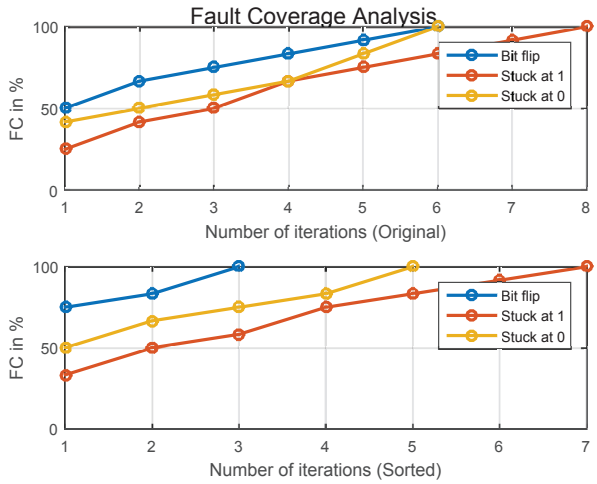

Fig. 5: Fault coverage analysis for different fault models on $C-17$ circuit

data is plotted for the dynamic compaction, whereas the sorted data is plotted after the static compaction. For that purpose, two parameters, namely compaction and hardness are calculated for different fault models. These parameters are described below.

A. Compaction

Compaction is defined as the minimum number of vectors obtained in order to reach the maximum FC of a circuit. The compactness of the $C-17$ benchmark circuit under analysis results in a different number of compact test vectors for different fault models. Based on this analysis, the performance of different fault models, which are used for the various combinational circuits, can be analyzed. Fig. 6 shows the compactness of different fault models for the $C-17$ circuits. The minimum number of test vectors is obtained for the experiment after the static compaction shown as sorted in the figure. A higher value of compactness gives a smaller number of compact test vectors [32]. The compactness is calculated

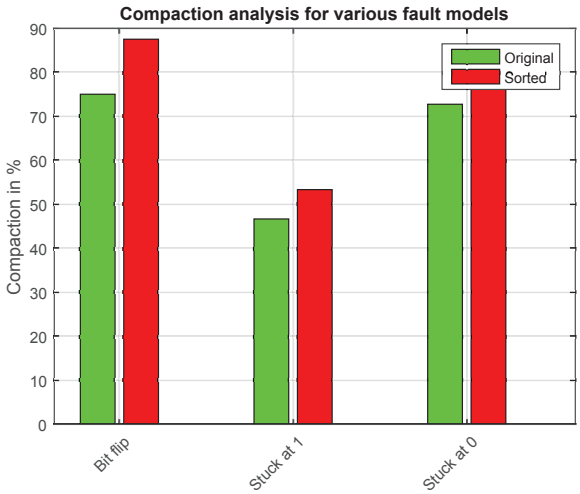

Fig. 6: Compaction achieved for different fault models

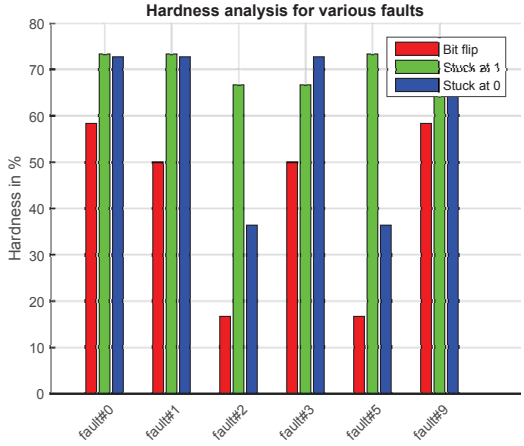

Fig. 7: Hardness for different fault diagnosis using threshold logic

mathematically as,

$$C = \left(1 - \frac{T_{\text{compact}}}{T_{\text{test\_vector}}}\right) \times 100\% \quad (3)$$

where C, T_{compact} and $T_{\text{test\_vector}}$ are compactness, minimum number of test vectors and total number of test vectors, respectively.

B. Hardness

Hardness or Hard to Detect (HTD) is the characteristic of those faults which can be detected very rarely. In the experiment, the occurrence or detection of faults are counted for all test vectors, which are obtained after the dynamic compaction. In the matrix, the sum of individual columns of the fault array represents the number of detections of individual faults. The HTD or hardness can be calculated as,

$$H = \left(1 - \frac{F_{\text{detections}}}{T_{\text{test\_vector}}}\right) \times 100\% \quad (4)$$

where H and $F_{\text{detections}}$ represent hardness and the number of detections of a fault for all test vectors, respectively.

Hardness of individual faults for both of the fault models is shown in Fig.7. If the hardness of a fault results in 100%, it means the fault is not detectable for any input, hence it can be called an untestable or undetectable fault. On the other side, a hardness of 0% shows the detection of fault for all test vectors, which means that the portion of the circuit where the fault has occurred is very sensitive to fault attacks. Thus, from this parameter, we can analyze the sensitive parts of the overall design. A threshold level is defined as a level that serves to illustrate whether the hardness of a particular fault of all fault models is below that threshold. if this is the case, any redundancy technique can be applied (e.g. Triple Modular Redundancy (TMR)) on only those locations in order to increase fault tolerance and robustness of the SUT. When there are many HTD faults, the number of compact test vectors will be higher than that of those having less hardness. In the experiment, a total of 12 faults namely fault#0,fault#1,...,fault#11 are injected. According to the threshold level, some fault locations have found which are very sensitive to the fault occurrence. In Fig. 7, the faults fault#0,1,2,3,5,9 represent the most frequent faults of all fault models. These fault locations are more sensitive to stuck at 1/0 and bit flip fault models.

TABLE II: Comparison between state of the art and this work

SUT	Patterns [31]	Patterns [33]	This work		
			Bitflip	*SA-1*	*SA-0*
C17	-	6	2	5	7
C432	32	57	28	42	59
C499	52	67	11	14	42
C880	25	66	30	47	63
C1355	84	96	28	39	39
C1908	-	142	53	40	50

This work is in progress and a unique methodology based on hardness of faults and fault locations is being developed in order to increase fault tolerance and robustness of the SUTs.

C. Result and Discussion

The main objectives of this paper are to generate high quality compact vectors for maximum FC and to develop a method of injecting faults in SUT written in the Verilog HDL for the FPGA based designs. Table II shows the comparison between the work presented in [31] and [33] and the approach presented in this paper. In [31], this work is compared with *Fastscan*, whereas in [33] different tools, e.g. PODEM, COMPACT, ATALANTA and SAT based approaches, are compared. Therefore, this proposed approach is compared with all of these tools. In these papers, stuck at fault model is used, whereas this work is presented on both bit flip and stuck at fault models. The different combinational circuits from ISCAS's 85 are compared and results are reported in table II. The table contains only those circuits for which comparison is available in the state of the art. Compact test patterns have obtained for various fault models individually. The FC values are found to be in range from 87% to 100% in majority of circuits, whereas the set point values are considered up to 30% to 50% of the total faults for different fault models.

VI. Conclusion

In this paper, an ATPG method is proposed with a simple hybrid compaction scheme. This work is compared with the state-of-the-art methods, and it is considered to be better in terms of the number of test vectors obtained and efficient memory utilization. In this methodology, faults are injected in a higher level of abstraction of the design process, i.e. in the Verilog code automatically. Three fault models are tested and patterns are found for them individually. The hardness of faults is also calculated, and the most sensitive fault locations for all fault models are diagnosed. This information is helpful for increasing the fault tolerance and robustness capability of the circuit. In future, this proposed method will be tested for sequential circuits and System on Chip (SoC) designs. The work on the hardness analysis is also in progress and the resulting information will be used to developed a unique redundant technique that helps to increase the robustness and the fault tolerance capability of FPGA based system designs.

REFERENCES

[1] M. Ahmady and S. M. Sayedi, "Fault Coverage Improvement and Test Vector Generation for Combinational Circuits using Spectral Analysis," in *25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, (Montreal, QC), pp. 1–5, IEEE, 2012.

[2] M. S. Sahari, A. K. A'ain, and I. Grout, "A study on the effect of test vector randomness on test length and its fault coverage," in *10th IEEE International Conference on Semiconductor Electronics (ICSE)*, (Kuala Lumpur, Malaysia), pp. 503–506, Ieee, sep 2012.

[3] A. Baid and A. K. Srivastava, "Generating test patterns for fault detection in combinational circuits using genetic algorithm," in *2013 Students Conference on Engineering and Systems (SCES)*, (Allahabad, India), pp. 1–4, IEEE, apr 2013.

[4] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, 2002.

[5] Z. Navabi, *Digital System Test and Testable Design Using HDL Models and Architectures*. Worcester, MA USA: Springer New York Dordrecht Heidelberg London, 2010.

[6] R. Dobai and E. Gramatová, "Test Pattern Generation for the Combinational Representation of Asynchronous Circuits," in *IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, (Vienna), pp. 323–328, Ieee, 2010.

[7] M. H. Schulz and E. Auth, "ESSENTIAL: An efficient self-learning test pattern generation algorithm for sequential circuits," in *International Test Conference*, (Washington, DC), pp. 28–37, IEEE, 1989.

[8] A. Czutro, S. M. Reddy, I. Polian, and B. Becker, "SAT-Based Test Pattern Generation with Improved Dynamic Compaction," in *27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, (Mumbai, India), pp. 56–61, IEEE, 2014.

[9] R. Drechsler, M. Diepenbeck, S. Eggersglüß, and R. Wille, "PASSAT 2.0: A Multi-functional SAT-based Testing Framework," in *14th Latin American Test Workshop (LATW),*, (Cordoba), p. 4799, IEEE, 2013.

[10] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, (San Jose, CA), pp. 85–90, IEEE, 2013.

[11] S. Eggersglüß, M. Yilmaz, and K. Chakrabarty, "Robust timing-aware test generation using pseudo-boolean optimization," in *IEEE 21st Asian Test Symposium*, (Niigata), pp. 290–295, IEEE, 2012.

[12] Eggersglüß Stephan, K. Schmitz, R. Krenz-Baath, and R. Drechsler, "Optimization-based Multiple Target Test Generation for Highly Compacted Test Sets," in *19th IEEE European Test Symposium (ETS)*, (Paderborn, Germany), pp. 1–6, IEEE, 2014.

[13] A. Czutro, I. Polian, P. Engelke, S. M. Reddy, and B. Becker, "Dynamic Compaction in SAT-Based ATPG," in *Asian Test Symposium*, (Taichung), pp. 187–190, IEEE, 2009.

[14] D. Xiang, W. Sui, B. Yin, and K.-t. Cheng, "Compact Test Generation With an Influence Input Measure for Launch-On-Capture Transition Fault Testing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 1968–1979, sep 2014.

[15] A. Raghunathan and S. T. Chakradhar, "Acceleration techniques for dynamic vector compaction," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, (San Jose, CA, USA), pp. 310–317, IEEE, 1995.

[16] I. Pomeranz and S. M. Reddy, "Dynamic test compaction for synchronous sequential circuits using static compaction techniques," in *Proceedings of Annual Symposium on Fault Tolerant Computing*, (Sendai), pp. 53–61, IEEE, 1996.

[17] T. J. Lambert and K. K. Saluja, "Methods for Dynamic Test Vector Compaction in Sequential Test Generation," in *9th International Conference on VLSl Design*, (Bangalore), pp. 166–169, IEEE, 1996.

[18] S. Jha, K. Chandrasekar, W. Wu, R. Sharma, S. Sengupta, and S. M. Reddy, "A Cube-Aware Compaction Method for Scan ATPG," in *27th International Conference on VLSI Design and 13th International Conference on Embedded Systems*, (Mumbai, India), pp. 98–103, IEEE, 2014.

[19] S. Jha, *Compaction mechanism to reduce test pattern counts and segmented delay fault testing for path delay faults*. PhD thesis, University of Iowa, 2013.

[20] S. T. Chakradhar and A. Raghunathan, "Bottleneck removal algorithm for dynamic compaction in sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 10, pp. 1157–1172, 1997.

[21] A. El-Maleh, S. Khursheed, and S. Sait, "Efficient Static Compaction Techniques for Sequential Circuits Based on Reverse Order Restoration and Test Relaxation," in *14th Asian Test Symposium (ATS'05)*, pp. 378–385, IEEE, 2005.

[22] M. Dimopoulos and P. Linardis, "Efficient static compaction of test sequence sets through the application of set covering techniques," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, pp. 194–199, IEEE Comput. Soc, 2004.

[23] M. S. Shirazi, B. Morris, and H. Selvaraj, "Fast FPGA-based fault injection tool for embedded processors," in *International Symposium on Quality Electronic Design (ISQED)*, (Santa Clara, CA), pp. 476–480, IEEE, 2013.

[24] A. R. Khatri, M. Milde, A. Hayek, and J. Börcsök, "Instrumentation Technique for FPGA based Fault Injection Tool," in *5th International Conference on Design and Product Development (ICDPD '14)*, (Istanbul, Turkey), pp. 68–74, 2014.

[25] M. Shokrolah-Shirazi and S. G. Miremadi, "FPGA-Based Fault Injection into Synthesizable Verilog HDL Models," in *2008 Second International Conference on Secure System Integration and Reliability Improvement*, (Yokohama), pp. 143–149, IEEE, 2008.

[26] K. Hirofumi Yonetoku, "Method of Generating a Pattern for Testing a Logic Circuit and Apparatus for Doing the Same," 2002.

[27] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, "On acceleration of SAT-based ATPG for industrial designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1329–1333, 2008.

[28] C. Bernardeschi, L. Cassano, M. G. C. a. Cimino, and A. Domenici, "GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1243–1254, 2013.

[29] L. Zaourar, Y. Kieffer, and A. Wenzel, "A multi-objective optimization for memory BIST sharing using a genetic algorithm," in *IEEE 17th International On-Line Testing Symposium*, (Athens), pp. 73–78, IEEE, 2011.

[30] Z. Ebadi and A. Ivanov, "Design of an optimal test access architecture using a genetic algorithm," in *Proceedings 10th Asian Test Symposium*, (Kyoto), pp. 205–210, 2001.

[31] C. Dunbar and K. Nepal, "Using platform FPGAs for fault emulation and test-set generation to detect stuck-at faults," *Journal of Computers*, vol. 6, no. 11, pp. 2335–2344, 2011.

[32] A. C. Abdullah and C. Y. Ooi, "Study on Test Compaction in High-Level Automatic Test Pattern Generation (ATPG) Platform," *Circuits and Systems*, vol. 4, no. 4, pp. 342–349, 2013.

[33] K. Habib, M. Safar, M. Dessouky, and A. Salem, "Don't Cares based Dynamic Test Vector Compaction in SAT-ATPG," in *IEEE 57th International Midwest Symposium Circuits and Systems (MWSCAS)*, (College Station, TX), pp. 213–217, IEEE, 2014.