 doi:10.20944/preprints202107.0403.v1

Article

Optimal reduction of number of test vectors for soft processor cores implemented in FPGA

Mariusz Wegrzyn [1], Ernest Jamro [2], Agnieszka Dąbrowska-Boruch [2] and Kazimierz Wiatr [2,*]

[1] Technical University of Cracow mariusz.wegrzyn@pk.edu.pl
[2] AGH University of Science and Technology {jamro, adabrow, wiatr} @agh.edu.pl
* Correspondence: jamro@agh.edu.pl

Abstract: This paper describes a new optimization methodology of testing vector sets reduction for testing of soft-processor cores and their individual blocks. The deterministic test vectors both for whole core and its individual blocks are investigated that significantly reduce the testing time and amount of test data that needs to be stored on the tester memory.
The processor executes an assembler program which together with determined testing vectors exercise its functionality.
The new BIST methodology applicable at industrial testing of processor cores, diagnostics and dynamic reconfiguration of FPGA is proposed. This novel methodology combined with dynamic reconfiguration of FPGAs can be profitable applied for missions-critical i.e. FPGAs operate in space, or other difficult condition where are explore on radiation. Experimental results demonstrate that the proposed approach reduces many times testing time.

Keywords: Processors testing; FPGA; test optimization

1. Introduction

Intensive development of the FPGA platform makes possible of leading research upon optimization on SoC designing, testing and implementation optimization of many algorithms and research experiments.

Now a day, FPGAs are designed for high level programing, application of Artificial Intelligence built-in engines, neural networks implementation. These resources can be applied to efficient implementation of BIST. On the other hand, FPGA based applications include usually higher number of processor cores implemented in, as application processors, real time microcontrollers. One can observe, that to testing of such cores are proposed mainly random-based testing methodologies, which requires huge number of testing vectors, advanced optimization algorithms and FPGA resources for their implementation.

For such a reasons methods of test vectors compression are developed in other to save memory resources to store them [1]. Pseudo-random stimuli generation is defined in the System Verilog HDL language standard [2], in the Universal Verification Methodology (UVM) [3]. Often various pseudo-random stimuli generators (PRGs) are utilized for this purpose. Such PRGs can be inbuilt in RTL simulators or external ones written in C++, and connected through direct programming interface [4].

The efficiency of stimuli generation is usually measured by coverage of injected faults, where authors mainly utilizes well-known ,,stuck-at" fault models.
Proposed in bibliography pseudo-random test-pattern generators are often realized as feedback-controlled. Such methods based on coverage analysis are called Coverage-Driven Verification (CDV). A drawback of this approach can be redundant number of random test vectors, if it happens, that coverage feedback is not properly propagated to PRG and reflected by suitable constraints [4]. Related optimization techniques already appeared in works in order to overcome mentioned above difficulties. Other similar solution

 doi:10.20944/preprints202107.0403.v1

is described in [5]. The work introduced capability of on-the-fly constraint optimization and generation of optimal stimuli set. GA was integrated directly into the UVM verification environment with optimized values of GA parameters. Kitchen and Kuehlmann in [6] proposed a Pseudo-Random stimuli Generators (PRG) using hybrid constraint solver based on Markov-chain Monte Carlo methods, which dynamically controls PRG.

Two main approaches based on coverage analysis are known from bibliography: feedback-based CDV (FBCDV) and CDV. FBCDV is based on a feedback from coverage analysis and modification of the constraints to the PRG. Whereas, CDVBC approach is based on a generated external model of DUV which is used to generate stimuli designed to satisfy the intended coverage [4]. By analogy our optimization methodology presented in this paper is supported by feedback information about fault coverage. CDVBC based approaches commonly consist in transformation of coverage situation into Boolean logic (e.g. conjunctive normal form) and gaining the power of SAT solvers [7], [8].

Finally, there are FBCDV approaches based on genetic algorithms (GA). Authors of [8] applied GA for automated generation of stimuli-based on source code of specific software application. Naturally, such approach neglected all the details concerning the processor hardware irrelevant to the verified application.

Different FBCDV solutions utilizes genetic algorithms (GA). Application of GA automated generation of stimuli based on source code of specific software application is presented in [9]. In this approach, only processor hardware resources utilized by verified application are taken into consideration. Such a solution are additionally time-consuming. Whereas a lack of deterministic quick BISTs which can be applied to periodical on-line tests of embedded processors cores is observable in bibliography. This with cooperation with dynamic FPGA reconfiguration methodology constitute efficient and powerful reliability mechanism. Moreover such a solution is easy to implement.

2. Bijective Program and faults modeling

2.1. Bijective test program

We generate a test sequence that allows arbitrary situations that might occur in practice. This is accomplished by using a test sequence that explores the functionality of each individual instruction and is composed in such a way that it forms a sensitive path. This path can be executed more than once, each time with a different input vector [10], [11].

Although we have borrow the notion of a sensitive path from the automatic test-pattern-generation (ATPG) techniques [12], [13] in our case it has a slightly different meaning [10]. The path sensitization in conventional ATPG techniques for automatic test generation involves the generation of the path that is sensitive to the presence of a stuck-at fault and the justification of the values along the path by propagating signals back to the primary inputs.

According to our approach the fault detection is performed at the instruction level by a compact test program in which individual processor instructions are organized in such a sequence that the destination register operand of i-th instruction represents the source register operand of (i+1)-th instruction. In the test sequence, each processor instruction participates at least once. The principle of instruction sequencing is presented in Figure 1.

 doi:10.20944/preprints202107.0403.v1

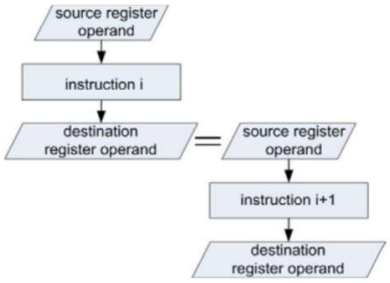


Figure 1. The principle of instruction sequencing.

We impose a stricter rule on the test-sequence generation by requiring that there is a one-to-one, i.e. bijective, correspondence between the input test pattern and the result.

Bijective function:

Definition: In mathematics [14], a bijection, **bijective function, or one-to-one correspondence** is a function between the elements of two sets: X, Y, where each element of set X is paired with exactly one element of set Y, and each element of set Y is paired with exactly one element of set X. There are no unpaired elements. Figure 2 presents bijective function.

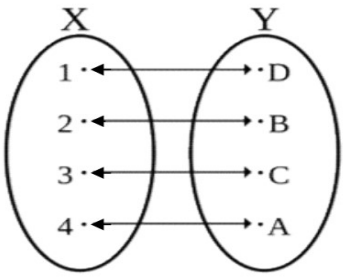


Figure 2. Bijection.

If we apply this rule at the level of sub-sequences of the assembler instruction sequence we can ensure high Fault Coverage (FC). The key achievement of earlier work [11] was proposed bijective testing procedure.

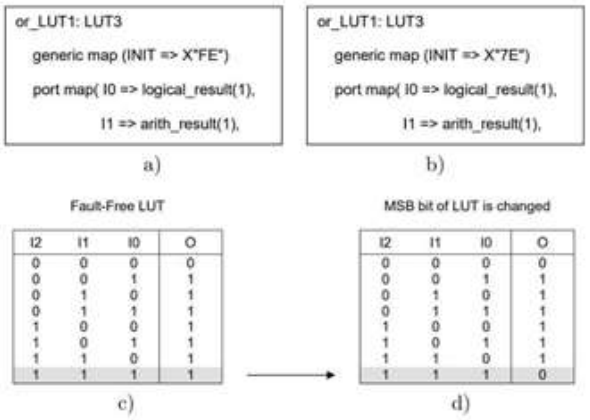
2. 2. **Faults Modelling**

In the proposed approach, the goal is to generate a compact test sequence that detects permanent SEU-induced faults of embedded processor cores in SRAM-based FPGAs [11]. As described in [15], the functional model of such faults differs considerably from the conventional stuck-at fault model due to the fact that SEU-induced faults affect logic elements implemented by the Look-Up Tables (LUTs), in this way that the logic function is arbitrarily changed. Permanent SEU-induced faults in LUTs are modeled by software injection in structural level of Hardware Description Language (HDL – in our case VHDL) description of targeted microprocessor.

An example [16] of a modelled fault is shown in Figure 3. The HDL description of a LUT implementing a three-input OR gate is shown in Figure 3a. and the corresponding truth table, in Figure 3c. The SEU-induced fault of a LUT typically manifests itself as a change of one bit of the LUT, thus modifying the Boolean function it implements. Let us assume that the most significant bit of the LUT has been corrupted, as shown in Figure

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

3d. The fault can be modeled by changing the initialization parameter (INIT), as shown in Figure 3b.



(a) HDL description of fault-free three-input OR gate,
(b) most significant bit of the LUT is changed (X"FE") → (X"7E")
Figure 3. Fault effect related to the change of one bit (X"FE") → (X"7E") in LUT3.

Our idea [11] is to use appropriate microprocessor simulator which accepts its specification in HDL, correlates it with the targeted FPGA, performs simulations with provided programs (in assembler) and allows analyzing behavior of the tested application (e.g. program results) in this environment. These simulations were performed by two simulators: Cadence NC VHDL and Mentor Graphics ModelSim. Fault injection is performed at microprocessor HDL structural description level, which reflects FPGA implementation.

The generation of the fault descriptions was implemented as a Perl script [11], [16]. All the instances of Look-Up-Tables LUTs contained in functional blocks of the processor are described in the VHDL code. For each LUT instance its initialization parameter is investigated and the list of the initialization parameters describing all the SEU-induced faults as well as all the stuck-at faults at the LUT inputs and outputs are generated. For some LUT instances it is possible that a single bit change of a LUT content manifests itself as a stuck-at fault. In such a case a duplicated stuck-at fault description is excluded. In a similar way the stuck-at faults at the LUT inputs as well as the stuck-at faults at the LUT output can also be modelled by modifying the contents of the LUT configuration. In some cases LUT SUE faults and stuck-at fault may result in the same LUT contents. In such a case, a duplicated fault description is omitted.

3. Optimal reduction of test vectors

One of the most important criterion of every test program evaluation are Fault Coverage (FC) and time required to complete. This time depends on both number of program instructions to be executed and number of applied test vectors. This paper focuses on optimization of the number of test vectors with as low as possible influence on FC. Such approaches may be especially profitable in the case of testing 32 or 64-bit microprocessors as there is a huge number of input test vectors required for exhaustive testing of these microprocessors. For instance there is 2^{32} possible input test vectors for a 32-bit microprocessor.

The superior objective is to find a minimal set of test vectors, which can achieve the maximal Fault Coverage (FCmax). This means, that developed optimization method should give the same FC as an exhaustive test. Another aspect of research is to select only a few vectors in such a way that the ratio of FC to the numbers of test vectors is optimal.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

A general test situation can be described by:

• the set of faults $F = \{f_1 , f_2 , ..., f_m \}$;
the set of available tests vectors $V = \{v_0 , v_1 , ...,v_{n-1} \}$, where vi corresponds to the execution of the test sequence (program) with a binary input value i ($0 \le i \le n$-1); in the case of 8-bit microprocessor $n = 256$.

• the fault-detect matrix D of dimension $m \times n$ describes detect-ability of faults f_j by test vector vi , $0 \le i \le n - 1$
The element d_{ji} of matrix D is set to 1 ($d_{ji} = 1$) in case when the test vector v_i detects fault f_j, otherwise ($d_{ji} = 0$) In this particular case the number of different injected faults is m = 1603, the number of different test vectors $n = 256$.

In order to better understand optimization of the number of test vectors, we proposed the following definitions:

Definition 3.1: *The fault of i-th order is called a fault detected exclusively by i test vectors.*

Definition 3.2: *The vector of i-th order is called a vector, which detects at least one fault of i-th order and does not detect any fault of lower order than i.*

Consequently, the most difficult to detect faults, called further the hardest faults are the first-order faults, which are detected only by one test vector. In our practical case we have found 41 faults detected only by one vector. Table 1 presents statistics on faults order. Faults of these orders are present as the outcome from the faults simulation experiment.

Table 1. Statistics on faults order.

Order	1	15	67	71	72	78	80	87	92	99	125	127	128
# faults	41	1	1	1	1	1	1	1	1	1	1	2	110
Order	135	144	160	161	168	175	176	184	190	191	192	193	194
# faults	1	1	2	1	2	1	3	6	1	4	32	2	3
Order	195	196	199	200	204	206	208	209	215	216	217	219	222
# faults	2	1	1	4	1	1	4	1	1	4	1	1	1
Order	223	224	225	226	227	230	231	232	234	235	236	239	240
# faults	1	34	4	8	3	2	1	2	4	2	1	2	103
Order	241	242	243	248	249	250	251	252	253	254	255	256	
# faults	19	2	2	31	2	3	2	6	3	10	146	731	

There are 41 faults of first order, one fault of 15-*th* order, one fault of 67-*th* order, and so on. There is the highest number of 256-*th* order faults (731). It is worth to notice, that faults of higher-orders are usually covered by first order vectors. By definition, it holds for 256-*th* order faults. Experiments proved that this holds also for the 15-*th* order faults and every higher order faults.

Above statistics gives us information how efficient is bijective test program. Hence, if the number of low orders faults was high, and high number of the lowest-order vectors was required to detect them, it would mean that small number of sensitivity paths were activated. Then it seems a good idea to improve the test program (written in PicoBlaze assembler).

Based on the above results, vector selection algorithms may be proposed in order to minimize the set of test vectors required to obtain maximal fault coverage (FCmax). At first greedy algorithm is proposed. This algorithm selects first the best vector, i.e. a vector which covers the largest number of faults.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

Algorithm 1: Greedy algorithm: the vectors that detect the largest number of faults first

Determine set F of all faults f_i
While F is not empty
{ Determine test vector v_i which covers maximum number of faults in set F;
Test the microprocessor with test vector v_i;
Remove all faults f_j that are covered by test vector v_i from set F;
}

In case of Algorithm 1 and penultimate version of the bijective program, the set of 33 such vectors appears enough to reach FCmax of 85.6% (see Figure 4). This experiment showed that application of all 256 tests vectors (exhaustive test) for the PicoBlaze processor is redundant. Moreover, it turns out, that we can shorten about eight times the exhaustive testing time.

In order to further reduce the number of test vectors without decreasing the FC, Algorithm 2 was proposed. This algorithm selects the lowest-order test vectors first.

Algorithm 2: the lowest-order vector first:
Determine set F of all faults f_i;
While F is not empty
{Select the lowest order fault f_i of set F;
Select a test vector v_i that detects fault f_i;
Test the microprocessor with test vector v_i;
Remove all faults f_j covered by test vector v_i from set F;
}

In the implementation of the above algorithm, 28 vectors are enough to obtain maximal fault coverage FCmax. One of the vectors (7D), has detected 12 first-order faults. Vector 7E has detected 3 of first-order faults. The other 26 vectors have detected only one first-order faults each (see Table 2).

Table 2. List of the first order vectors.

# iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Test vector	7D	7E	FB	FA	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
# detected 1-st order faults	12	3	1	1	1	1	1	1	1	1	1	1	1	1
# iteration	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Test vector	EE	DE	CE	BE	AE	9E	FC	6D	5D	4D	3D	2D	1D	77
#detected 1-st order faults	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Initially it might seem that greedy algorithm (Algorithm 1) should give a better result (lower number of test vectors) than Algorithm 2. However, it is not the case, Algorithm 2 results in 28 test vectors, in comparison to 33 test vectors for Algorithm 1. After thorough consideration, it is obvious that starting with first-order test vectors gives the optimal solution as in order to obtain FCmax all first-order vectors must be tested. This can be easily derived from Definition 3.1 and 3.2.

 doi:10.20944/preprints202107.0403.v1

Nevertheless it is not obvious, how Algorithm 2 should be constructed after all first-order vectors have been tested. Fortunately in our case, testing all first-order vectors is enough to cover all higher order faults. Based on statistics of fault orders and further experiments, we can conclude, that most of higher order faults are covered by many vectors from the set of the first order. Hence, we may propose the method of testing the processor consequently with increased order vectors, until all faults are covered. Nevertheless in general case, Algorithm 2 should be improved. When there are two or more vectors of the same lowest order, several different vectors can be taken. In this case the proposed Algorithm 3 may be used. For Algorithm 3 the vector which covers the largest number of the lowest order faults is selected. Consequently, improved algorithm, Algorithm 3 - Hybrid, is a mixture of Algorithm 2 and 1, however Algorithm 2 is higher priority algorithm. Algorithm 3 is especially useful in the case when the lowest-order vector is a second or larger order one, as in this case there are two or more vectors that cover the same fault. A proposition of the improvement is given below. It should be noted that performance of Algorithm 3 was not tested in practice as in this experiment testing only first-order vectors result in FCmax.

Algorithm 3: Hybrid (improved the lowest-order vector first)

Determine set F of all faults f_i;
While F is not empty
{Determine subset F_i of F with the same, lowest-order faults f_i;
Select a vector v_i that detects the largest number of faults from set F_i;
test the microprocessor with test vector v_i;
from set F, remove all faults f_j that are covered by test vector v_i;
}

In our case all faults are covered by the first order vectors, therefore Algorithm 3 and Algorithm 2 require the same number of vectors to obtain FCmax. Nevertheless, in general case, Algorithm 3 should require less testing vectors. On the other hand Algorithm 2 is simpler and requires slightly less calculation time. Algorithm 1 - ,,Greedy" requires higher number of iterations to achieve FCmax. Algorithm 3 ,,Hybrid" turned out to be the best in this practical case.

4. Further reduction of test vectors number

In our case testing all vectors (or selected 28 vectors) gives a Fault Coverage at the level of 85.6 % which is denoted as FCmax. It should be noted that some of these undetected faults cannot be detected at all due to e.g. logic redundancy. Some faults are so hard to detect that only specific instructions with specific input vectors and defined processor states are affected. In practice these instructions are in most cases not used.

In some cases we want to further reduce the number of test vectors at the cost of lower FC. At practice 95 or 97 % of FCmax may be satisfactory. For this reason we have checked, how quickly the FC tends to FCmax, applying presented before algorithms.

The initial goal was to reach FCmax with the lowest number of vectors (the result was 33 vectors), which is larger in case of Algorithm 1 than for Algorithm 2 (28 vectors). However, the greedy algorithm (Algorithm 1) results in the fastest increase of FC for the initial iterations. This holds by the definition, the greedy algorithm takes the best possible vector at each iteration (but no global optimization is used) therefore a different algorithm (global optimization) may give a better result only after two or more iterations.

Algorithm 2 requires 28 first order test vectors to achieve FCmax. The FC achieved when one of these vectors was applied alone is given in Table 3. Based on these results the order of applied test vectors is determined at the very beginning of this algorithm (before

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

Algorithm 2 is started). This order had been determined once. However this approach allows to achieve FCmax, but the number of iterations required to achieve 97% of FCmax is optimized further in Section 5.

Table 3. First-order vectors and their FC.

Vector	F2	F0	F4	F6	F1	FA	F8	EE	CE	2D
Detected faults	1249	1240	1240	1236	1231	1229	1227	1226	1221	1220
%FC max	91,03	90,38	90,38	90,09	89,72	89,58	89,43	89,36	88,99	88,92
Vector	FC	F5	F3	77	1D	DE	F9	F7	AE	3D
Detected faults	1220	1218	1217	1214	1209	1205	1203	1202	1198	1193
% FCmax	88,92	88,78	88,70	88,48	88,12	87,83	87,68	87,61	87,32	86,95
Vector	9E	FB	4D	BE	5D	6D	7E	7D		
Detected faults	1187	1185	1184	1175	1164	1157	1149	1099		
% FCmax	86,52	86,37	86,30	85,64	84,84	84,33	83,75	80,10		

5. Aggregated FC vs number of test vectors

The comparison of the aggregated FC for all three Algorithms is shown in Figure 4. Algorithm 1 results in the highest FC only for three initial iterations. Higher FC achieves Algorithm 3 since 4-th iteration. Using Algorithm 2, the FC increased irregularly. This algorithm is not optimal. However this algorithm is easier to implementation and quicker to execution than Algorithm 3. For Hybrid Algorithm, the number of covered faults are calculated at every iteration of the algorithm. Algorithm 1 ,,Greedy" can be used when the rapid increase of FC in the first few iterations is required. On the other hand, this algorithm requires the highest number of iterations (33) to complete (obtain FCmax). It is predictable, that for other sets of input data, the difference in the number of iterations may be even greater in favor of Algorithm 3 ,,Hybrid''. Taking into consideration both number of iteration required to achieve FCmax and FC increase rate, it is possible to conclude that Algorithm 3 is the best. However more sophisticated algorithms i.e. exhaustive search, genetic algorithm, simulated annealing might give better solutions.

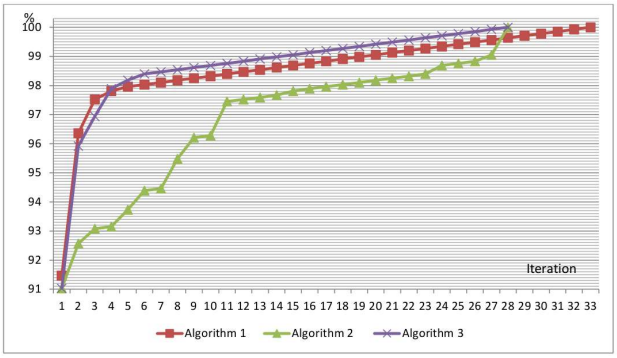


Figure 4. The comparison of the FC aggregation for all three Algorithms.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

6. Local test vectors

Up to now global test vectors have been only considered, i.e. a single input vector is applied for the whole test program. The global test vectors reduce the number of input / output data transfer between processor and external test control module, reduce the memory size and results checks. Nevertheless, when these factors are neglected, further reduction of test time might be obtained by employing local test vectors. A local vector is applied only for a single processor block (specified assembler instructions).

PicoBlaze VHDL description is divided into 13 blocks by its designer (Ken Chapman). We have determined optimal sets of test vectors separately for every hardware block of the processor. We utilized primarily the Algorithm 3 – Hybrid, and Algorithm 1 to this task. Algorithm 1 is applied usually for blocks, where the lowest order of test vector was relatively high, i.e. 67 or higher. The number of local test vectors required to achieve FCmax(block) and index of the algorithm, which gave the best result is presented in Table 4.

Table 4. Number of local test vector.

#	**Processor's block**	**# vectors**	**The fastest Algorithm**
1	Fundamental control	1	3
2	Interrupt (input, enable, flags)	1	3
3	Decodes for Control PC & CALL/RET stack	2	1, 3
4	The Zero, Carry flags	28 (1-order)	3
5	PC, Def. 10-bit counter	2	1, 3
6	Register bank and 2nd operand	2	1
7	Memory storing	1	3
8	Logical instructions	3	1
9	Shifts instructions	2	1, 3
10	Arithmetical instructions	3	1
11	ALU multiplexer	2	1
12	Read/write strobes	2	1, 3
13	Program CALL/RETURN stack	2	1, 3

Three blocks require only one test vector to obtain FCmax(block). Most blocks require two or maximum 3 vectors to obtain FCmax(block). Seven blocks require two such vectors. Only two blocks require three such vectors. However the worst block is Zero and Carry Flag block which requires as much as 28 vectors to obtain FCmax(block).
Furthermore, there exist two PicoBlaze HW blocks, where global vectors rather than local are required. Such blocks are ZERO and CARRY Flags, Program Counter (PC). Every kind of instructions such as logical, arithmetical and shifts have capability to generate flags. Every instruction of the bijective program tests PC indirectly.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

Percentage distribution of detected faults in every individual PicoBlaze block is presented in Figure 5. We can observe that the best results are achieved for blocks which operate directly on data (Blocks 6-10 in Table 6.9).

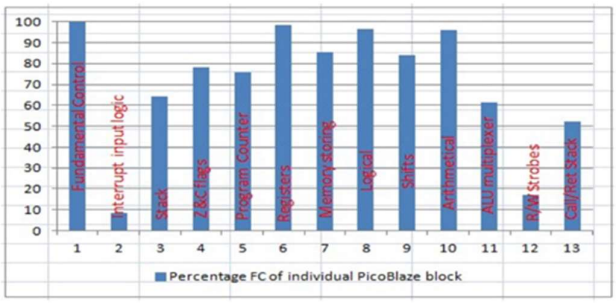


Figure 5. Percentage of FC of individual PicoBlaze block.

ALU Multiplexer is tested indirectly by assembler functions and in this block exists hardware redundancies. Moreover a lot of undetected faults have its place in HW which realizes I/O operations (about 39% injected in this block). For this reasons FC is here relatively low. The most difficult one to test is the ZERO and CARRY flags generation block. Hence, there is the higher number of first order faults and vectors to detect them. Hardware architecture of this block is most complex. Many one-bit details on both operand (256×256) are required to detect all possible situations related to faults in this block. Moreover a few instructions can generate the same flags in situation when all 8 bits are taken under calculations and range of register is limited to 7, 6, 5, etc. bits. FC for PicoBlaze blocks dedicated to Program Counter is low too, because testing of these processor resources is not main task of this bijective program, as mentioned above.

Unfortunately, Zero and Carry Flag block requires the maximum 28 test vectors. Therefore local vectors cannot in a direct way reduce test time. One of the solution to this problem might be to further optimize testing procedure that Zero and Carry Flag block has separate testing program. For example, we may design similar testing procedure only for the Zero and Carry Flag block, i.e. check how many vectors are required for each processor block when only flag block faults are injected. Drawback of this solution is that the program size will grow and testing procedure is complicated.

Another solution is to analyze an individual user's program for what kind of flag instructions are used. In most cases, the flag register is modified but the next instruction ignores the flag states; in most cases only branch instructions check the flag state. So only a few instructions need to be tested in the Zero and Carry flag.

7. Configuration Readback

Almost 100% FC can be obtained by using FPGA configuration readback, i.e. when configuration memory SUE error occurs we can read the configuration memory and compare it with the original one. Therefore, in theory, this results in 100% FC. Nevertheless, hard faults, i.e. stuck-at errors might be not detected by readback, as these faults are not associated with the FPGA configuration memory corruption. Nevertheless this kind of faults are less common than SEU. Such sorts of faults can be detected by our test program. Stuck-at faults at inputs and output of LUTs were modelled [11], [16].

The configuration readback method does not detect any application faults of data stored in BRAM. Fortunately, additional parity bits checking can easily detect SEU faults. SEU fault detection is much more complicated in the case of registers (standard or pipeline) and distributed RAM (scratchpad or stack memory). In this case, parity bit checking would require redesigning the processor core with associated arithmetic and logic modules. The alternative might be Triple Modular Redundancy (TMR). The recent research

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2021 doi:10.20944/preprints202107.0403.v1

revealed, that the method of partial reconfiguration assisted by TMR or testing achieves the best results. On the other hand, there are static FPGA resources, which cannot be partially reconfigured, such as global connections, logic of interfaces, and part of clock resources. The method of partial reconfiguration is usually complex, and requires three bitstreams (original, read-back, and masking data, that can change during operation). Xilinx elaborated a CAPTURE tool, which makes possible storing an application data before readback. Moreover testing by readback method requires knowledge about placement and utilization of elements (Frame Address Register).

Nevertheless TMR has its drawbacks too. Namely requires three times more hardware resources, moreover TMR is sensitive to higher number of faults, and may not satisfy its function in this case. One of most important TMR component, so called Majority Voter (MV) is the most critical circuit. For this reason newer and newer solutions of MV are still developed, both on logical and technological level. New TMR solution, more and more immune to SEU are also created as: Temporal, Partial or Partitioned TMR [17], [18], [19], [20].

Reading back whole FPGA configuration memory is time consuming – requires similar time to FPGA programming. Complete bitstream for Virtex Ultrascale+ VU3P contains 213'752'800 bits. Using SelectMAP mode or the ICAP, this BIT file could be loaded in about: 213'752'800 bits / 3'200'000'000 bps $\approx$ 66.79 milliseconds.

The time can be significantly reduced when partial configuration readback is employed. For example a small partial BIT file for a Virtex-7 device contains a region spanning 100 Slices. Before the rawbits (.rbt) file is generated, the configuration time can be estimated by using the bitstream size provided by the PlanAhead software.

Nowadays partial reconfiguration technique plays a key role for digital programmable systems, where high reliability is required. In high reliability systems, the partial reconfiguration is often supported by testing for mentioned above reasons. Testing according to our methodology under certain condition [11], [16] may require similar amount of time as partial reconfiguration. Table 5 presents a comparison of reconfiguration and testing times for PicoBlaze processor core. First column presents partial reconfiguration time with exclusively given processor core. Additionally fast version of PicoBlaze (PicoBlazeHZ) was taken into consideration.

Table 5. Testing time vs reconfiguration time.

PicoBlaze in Virtex-7 (XC7VH870T)				
Partial PB configuration time (50CLB)	370 instructions	3 vectors x 370 instructions	28 vectors (the worst case)	Full configuration readback
Processor clock 100 MHz				
36us	3,7 us	11,1 us	103,6 us	91,88 ms / 3,2 Gbps
Processor clock 240 MHz (PicoBlazeHZ)				
36 us / 3,2 Gbps	1,54 us	4,63	43,17	91,88 ms / 3,2 Gbps

There is not optimal solution for every application. In the case when fast run-time (less than roughly 10 μs) is required, the TMR seems the only solution. On the other hand, when accepted delay is more than 100 ms, configuration readback is the best solution. This readback should be associated with the proposed test and user's program execution two or three times in order to detect register / distributed memory SUE. In the case when accepted delay is between roughly 10 μs and 100 ms different solutions might be used. One of them is (partial) readback combine with the proposed testing solution. Another solution might be Dual Module Redundancy combine with the proposed testing procedure to select a proper result. The proposed testing procedure can also be adopted in the

case of TMR when two or more SUE occurs. In these cases selecting the number and / or value (order) of test vectors might be a very important issue and it is application dependent.

8. Conclusions

Testing FPGAs soft-processors are often neglected due to the fact that in most cases either configuration readback or Triple Module Redundancy (TMR) is used. Nevertheless, testing combine with readback or TMR might be still a good solution. Furthermore for allowed delay of 10 μs to 100 ms, soft-processor testing might be the very fundamental mission-critical procedure.

In this paper the SEU-induced fault model in FPGA was presented. Based on the model and the bijective testing procedure, an automated tool was designed to construct the fault-detect matrix *D*. This matrix specifies faults coverage for every possible test vector. Based on the matrix *D*, three novel optimization algorithms: Algorithm 1 (greedy), Algorithm 2 – (lowest-order first) and Algorithm 3 – (hybrid) were developed in order to reduce the number of required test vectors without reducing the obtained fault coverage (FC). In the given case study, the number of required global test vectors is reduced from 256 (8-bit microprocessor) to 31 for (Algorithm 1) or even to 28 for Algorithm2 or Algorithm 3. By the introduced theory, it is proved that Algorithm 2 obtains optimal number of test vectors provided that only first-order vectors are used to obtain FCmax. In a general case, when second or higher order vectors are used, Algorithm 3 seems to be the best, however it was not proved in practice, as in our case, testing only first-order vectors results in the FCmax. It might be even possibly to use Algorithm 2 for first order vectors and an Exhaustive Search Algorithm for second and higher order vectors.

In some cases testing time is limited and thus the number of test vectors need to be further reduced sacrificing the level of fault coverage. This case has been also studied, as the result in our case, the greedy algorithm is the best when only three vectors are tested. Otherwise Algorithm 3 should be employed. This hybrid algorithm considers both global and local optimisation. It should be noted that testing only three vectors results in more than 97% of FCmax.

Further reduction of number of testing vectors might be obtained by employing local test vectors. These test vectors are used only for a specific microprocessor block. Most blocks can be fully tested by only two or three vectors. Unfortunately, 28 vectors are required to test carry and zero flag generation block, and the flag register is influenced by most instructions. It should be noted that the carry flag register is, in most cases, used only for branches. Therefore further reduction of the number of test vectors can be obtained by analysing an individual program. The drawback of local vector usage is that more input and output vectors should be transferred to / from the microprocessor, and in some cases these vectors transfer may be more problematic than increased test run-time in the case of global vectors. Further optimization may be achieved by employing a hybrid method: global and local vectors usage, i.e. employing two global vectors and 26 local vectors to test only the flag register block.

Authors Contribution: Conceptualization, M.W., E.J., A.D.-B.; Methodology, M.W., E.J., A.D.-B., K.W.; Software, M.W.; Validation, M.W., E.J., A.D.-B.; Formal Analysis, M.W., E.J., A.D.-B., K.W.; Investigation, M.W. E.J.; Resources, M.W.; Writing – Original Draft Preparation, M.W., E.J.; Writing – Review & Editing, M.W., E.J., A.D.-B., K.W.; Visualization, M.W., E.J., A.D.-B.; Supervision, E.J., K.W.; Project Administration, E.J., K.W.; Funding Acquisition, K.W.

References

1. Kedarnath, J. B. and Nur, A. T. Matrix-Based Test Vector Decompression Using an Embedded Processor. *Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems* **(DFT'02)**, pp. 1063-6722/02
2. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language **2013**, *IEEE Std 1800-2012* (Revision of IEEE Std 1800-2009).

3. *Universal Verification Methodology (UVM) 1.2 Users Guide*, Accellera Systems Initiative **2015.**
4. Fajcik, M.; Smrz P.; Zachariasova M. Automation of Processor Verification Using Recurrent Neural Networks. *18th International Workshop on Microprocessor and SoC test and Verification (MTV)* **2018.**
5. Simkova, M. and Kotasek Z., Automation and Optimization of Coverage-driven Verification. *Euromicro Conference on Digital System Design*, Aug **2015**, pp. 87–94.
6. Kitchen, N. and Kuehlmann A. Stimulus generation for constrained random simulation. *IEEE/ACM International Conference on Computer-Aided Design*, Nov **2007**, pp. 258–265.
7. Yeh, H. and Huang, C., J. Automatic Constraint Generation for guided random simulation. *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan **2010**, pp. 613–618.
8. Cheng, A. C.; Yen, C. C. J.; Val, C. G.; Bayless, S.; Hu, A. J.; Jiang I. H. R. and Jou J. Y. Efficient Coverage-Driven Stimulus Generation Using Simultaneous SAT Solving, with Application to System Verilog. *ACM Trans. Des. Autom. Electron. Syst.*, **2014**, *vol. 20, no. 1,* pp. 7.1–7.23.
9. Goloubeva, O.; Reorda, M. S. and Violante, M. Automatic generation of validation stimuli for application-specific processors, *Proceedings Design, Automation and Test in Europe Conference and Exhibition,* Feb **2004**, *vol. 1,* pp. 188–193.
10. Wegrzyn, M.; Biasizzo, A.; Novak, F. Application-oriented testing of embedded processor cores implemented in FPGA circuits. *International Review on Computers and Software* **2007**, 2: 666-671.
11. Wegrzyn, M.; Biasizzo, A.; Novak, F.; Renovell, M. Functional Testing of Processor Cores in FPGA-Based Applications. *Computing and Informatics* **2009**, Vol. 28, 97-113.
12. Doumar, A.; Ito, H. Testing the logic cells and interconnects resources for FPGAs. *8 th Asian Test Symposium* **1999**, 369-374.
13. Renovell, M.; Portal, J. M.; Faure P. A Discusion on Test Pattern Generation for FPGA- Implemented Circuits. *Journal of Electronic Testing: Theory and Applications*, **2001**, 17: 283-290.
14. *Wikipedia 2020* https://en.wikipedia.org/wiki/Bijection
15. Safi, E.; Karimi, Z.; Abbaspour, M.; Navabi, M. Utilizing Various ADL Facetes for Instruction Level CPU Test. *Proceedings of the Fourth International Workshop Microprocessor, Test and Verification* **2003**, 38-45.
16. Wegrzyn, M.; Sosnowski, J. Tracing Fault Effects in FPGA Systems. *INTERNATIONAL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS* **2014**, Vol.60, NO. 1, 103-108.
17. Katkoori S.; Islam S., A.; Kakarla S. Partial evaluation based triple modular redundancy for single event upset mitigation. *INTEGRATION, the VLSI journal 77*, **2021**, 167–179.
18. Sielewicz, K. M.; Rinella, G. A.; Bonora, M.; Giubilato, P.; Lupi, M.; Rossewij, M. J.; Schambach J. and Tomas Vanat, T. Experimental methods and results for the evaluation of triple modular redundancy SEU mitigation techniques with the Xilinx Kintex-7 FPGA. IEEE Radiation Effects Data Workshop (REDW) **2017.**
19. Cui, X.; Liansheng L. Mitigating single event upset of FPGA for the onboard bus control of satellite. *Microelectronics Reliability 114,* **2020**, 113779 2020
20. Bolchini, C.; Miele, A.; Santambrogio, M. D. TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs. *22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems* **2007**.