RASP-TMR: An Automatic and Fast Synthesizable Verilog Code Generator Tool for the Implementation and Evaluation of TMR Approach

Abdul Rafay Khatri, Ali Hayek, and Josef Börcsök
Department of Computer Architecture and System Programming,
University of Kassel, Kassel, Germany

***Abstract*—Triple Modular Redundancy (TMR) technique is one of the most well-known techniques for error masking and Single Event Effects (SEE) protection for the FPGA designs. These FPGA designs are mostly expressed in hardware description languages, such as Verilog and VHDL. The TMR technique involves triplication of the design module and adding the majority voter circuit for each output port. Building this triplication scheme is a non-trivial task and requires a lot of time and effort to alter the code of the design. In this paper, the RASP-TMR tool is developed and presented that has functionalities to take a synthesizable Verilog design file as an input, parse the design and triplicate it. The tool also generates a top-level module in which all three modules are instantiated and finally adds the proposed majority voter circuit. This tool, with its graphical user interface, is implemented in MATLAB. The tool is simple, fast and user-friendly. The tool generates the synthesizable design that facilitates the user to evaluate and verify the TMR design for FPGA-based systems. A simulation scenario is created using Xilinx ISE tools and ISim simulator. Different fault models are examined during simulations such as bit-flip and stuck at 1/0. The results using various benchmark designs demonstrate that the tool produces synthesizable code and the proposed majority voter logic perfectly masks the error/failure.**

Keywords*—Fault injection; fault tolerance; reliability; single event effects; triple modular redundancy; Verilog HDL

I. Introduction

The Field Programmable Gate Array (FPGA) has been a widely accepted solution in developing the embedded system during the last few decades. Owing to its remarkable features such as parallelism, reconfiguration, separation of functions, self-healing capabilities, and overall availability [1], the FPGA has become the core of many embedded applications. The major applications include aerospace, biomedical instrumentation, safety-critical systems, spacecraft, Internet of Things (IoT), and many others [2], [3]. However, FPGA-based devices are susceptible to Single Event Effects (SEE) caused by various sources such as, α-particles, cosmic rays, atmospheric neutrons, and heavy-ion radiations. Since the capacity of FPGA chip has been growing by reducing the size of components integrated on the chip. This makes the device more prone to SEEs which provoke Single Event Upsets (SEU) in memory elements and Single Event Transients (SET) in combinational logic elements [4], [5].

Several SEE mitigation techniques have been proposed in the literature to avoid the effects of such errors in FPGA-based designs [6]. The reliability of the FPGA systems is improved by various error mitigation schemes such as multiple-redundancy with voting, Triple Modular Redundancy (TMR), hardened memory cell level, and Error Detection And Correction (EDAC) coding. Among all SEU mitigation techniques, TMR has become the most common practice because of its straightforward implementation and achieved the reliable results [6], [7], [8], [9]. The TMR mitigation scheme uses three identical logic circuits for performing the same task in parallel with corresponding outputs obtained through majority voters. This technique is the simplest redundant technique conceptually and it was invented by Von Neumann [10] in the year 1956. There are certain merits and demerits of TMR technique:

- Merits:
  - Simple and straight forward approach.
  - Improves fault tolerance and reliability.
- Demerits:
  - Large area overhead, nearly 200%.
  - More power consumption.
  - More cost or uneconomical.

FPGA designs are written in Hardware Description Languages (HDL) which describes the designs in various abstraction style, for example, gate, data-flow and behavioural levels. For small designs, gate abstraction style is employed and testing & verification processes are directly and easily applied to the designs. At this level, designs look more similar to the actual hardware representation. Data-flow and behavioural abstraction styles are used to implement the large designs.

Building a triplication scheme into digital designs is not a simple task. It takes a lot of time to build and debug the system. It has been also a laborious and error-prone task. Xilinx Inc. with the help of Sandia lab developed Triple Modular Redundancy (XTMR) tool for the design triplication. XTMR works with HDL and synthesis tool to automatically built TMR methodology. It is designed for the SEU and SET immunity for the FPGA devices, which includes Virtex family of reconfigurable FPGAs [11]. It requires two files, with extensions *.ngc (generated by synthesis process) and *.ngo (generated after the NGDBuild process) files, as an input and generates the TMR logic for the design. This tool is vendor specific and only used for Virtex family of FPGA devices. The TMRG (Triple Module Redundancy Generator) tool is developed in Python at CERN research institute, which automates the process of triplication of digital circuits at implementation stage. It is used to mitigate the SEEs. TMRG works on Verilog code and adds the classical majority voter circuit in TMR approach [12]. Brigham Young University (BYU) developed BL-Tmr tool [13], which works on EDIF-format design. It parses input EDIF file(s) into a net-list data structure and uses the classification information to select circuit structures for triplication. This tool works for partial TMR approach. Another tool, named TLegUp, which automatically generates Triple Modular Redundant designs for FPGAs from C programs using high-level synthesis technology [14]. Recently, some commercial tools such as Synopsys Synplify Premier, and Mentor Precision HiRel are available in the market to implement TMR during the synthesis process [15]. Using commercial tools, the cost of the project increases unnecessarily.

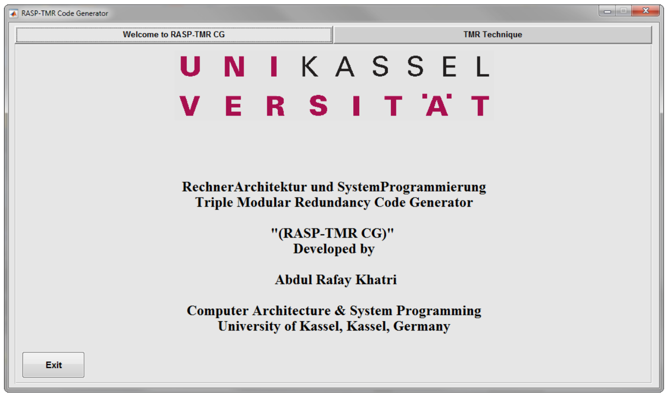
In this work, a tool named RASP-TMR Code Generator (RechnerArchitektur und SystemProgrammierung - Triple Modular Redundancy) is presented, of which the first part is the German name of our department. It takes Verilog HDL design file as an input and generates the synthesizable Verilog code for TMR technique. A new and simple majority voter circuit is also proposed. To validate the operation of the proposed tool, a simulation set-up is created with the help of Xilinx ISE tools and ISim simulator. The TMR operation is validated by injecting bit-flip and stuck at 1/0 faults in the design during the simulation, and it has been observed that the proposed majority voter circuit perfectly masks the errors/failures. This tool, along with its graphical user interface, is developed in MATLAB and it requires the users to provide only Verilog module file and then it automatically generates all the designs necessary to perform TMR. Hence, the RASP-TMR tool is simple, fast and user-friendly. To validate these claims, various benchmark designs are evaluated.

The structure of the paper is as follows: Section II explains the structure of the proposed RASP-TMR in detail, along with the proposed majority voter circuit. Section III presents the results of the RASP-TMR tool for combinational and sequential designs from ISCAS'85, ISCAS'89, and EPFL benchmark designs. Finally, Section IV concludes the paper.

II. Proposed RASP-TMR Code Generator

The RASP-TMR code generator tool, with its graphical user interface, is developed in MATLAB. It is a tabbed based tool and Fig. 1 shows these tabs. The first tab depicts the information about the tool, whereas, the second tab consists of the RASP-TMR tool. The tool consists of 9 functions and 254 lines of code in MATLAB. To provide ease of use, a standalone app is created using MATLAB command `deploytool`. This app is installed by the user on any host machine, having a Windows operating system. This tool not only triplicates the design and generates Verilog top file but also instantiates the designs in the top file. The proposed majority voter circuit is added for each output port of the design.

Fig. 2 depicts the flowchart of the RASP-TMR tool, which includes the series of operations performed by RASP-TMR code generator tool. The RASP-TMR tool accepts a Verilog design file as an input, parses it, obtains tokens (input arguments, output arguments) and makes three copies of it. Module name and the output argument must be changed in order to differentiate from each other and brought them under



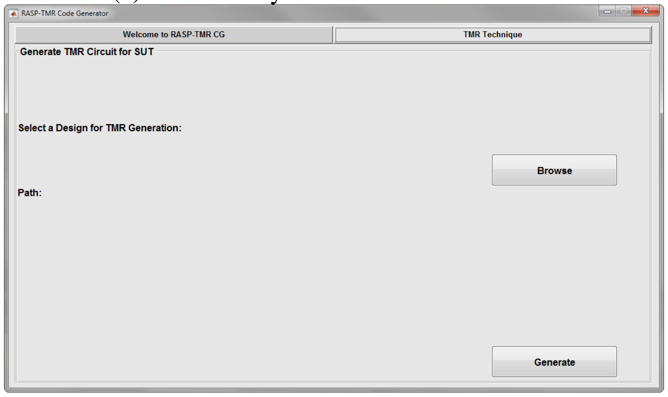
RechnerArchitektur und SystemProgrammierung
Triple Modular Redundancy Code Generator

"(RASP-TMR CG)"
Developed by

Abdul Rafay Khatri

Computer Architecture & System Programming
University of Kassel, Kassel, Germany

(a) Introductory tab of RASP-TMR tool.



(b) RASP-TMR code generator tab.

Fig. 1. GUI of the proposed RASP-TMR code generator.

the top-level file. RASP-TMR also generates the top-level module file which includes input arguments, output arguments, instantiation of all three TMR modules and proposed majority voter circuit for each output port.

A. Top File Structure

The top file consists of different components such as triplication of the module, their instantiations, and a proposed majority voter circuit. Fig. 3 shows the general block diagram of the top file generated by the RASP-TMR. These components are described in the sequel.

1) Triplication of Module: The triple modular redundancy requires the triplication of a module. All three modules operate in parallel. If any of a module fails to perform the intended task or results in an error, it is masked by the majority voter circuit. Therefore, when a module is input to the RASP-TMR code generator tool by the user, it parses the design and triplicates it. It should be noted that the module name changes from c17 to c17_1, which denotes the first module of TMR designs. The other two modules are renamed to c17_2 and c17_3 respectively. The output ports are changed as shown in Fig. 4. It shows the original Verilog design as an example which is input to the RASP-TMR tool (above) and the output of the tool (below).

2) Instantiation Generator: Xilinx ISE design tools provide the built-in instantiation generator for modules available in the design to instantiate one module within another thus creating hierarchy. RASP-TMR has the ability to instantiate the

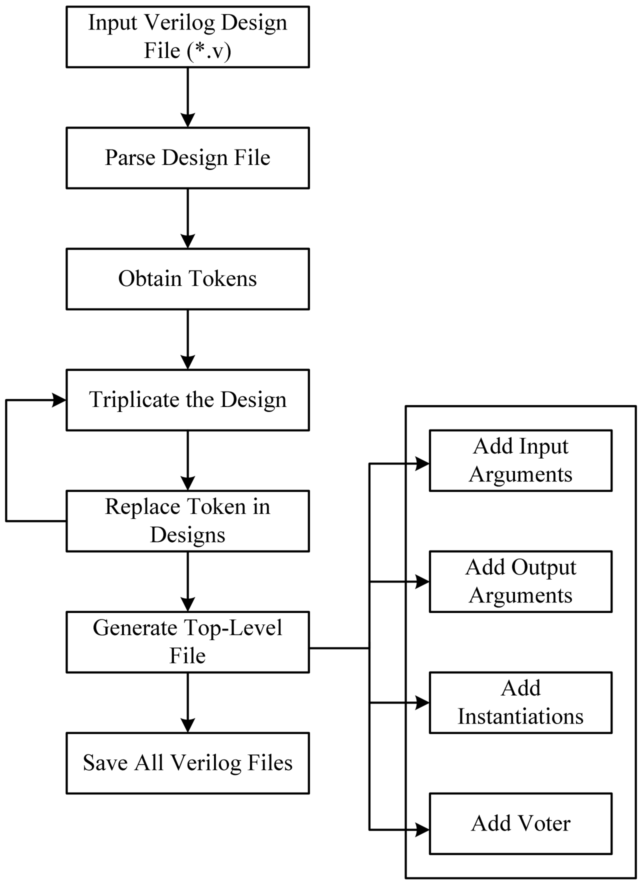


Fig. 2. Flow chart of the RASP-TMR tool.

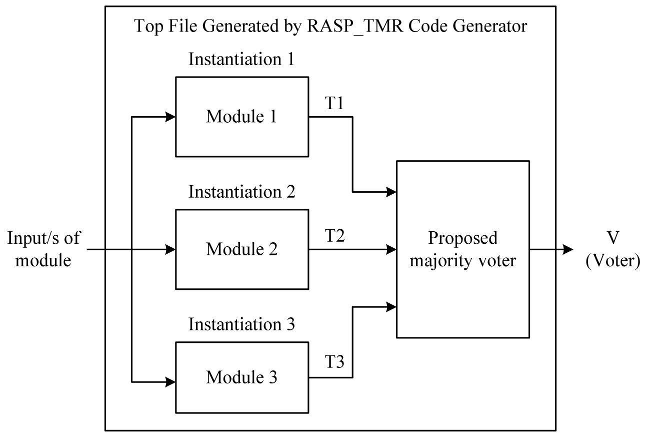


Fig. 3. Structure of top file generated by RASP-TMR tool.

TMR modules in the top file. The function `InstanceGen()` is developed under RASP-TMR tool which generates the instantiations. Instantiation requires the module name, produces instance name and adds the information about the input/output ports. Fig. 5 shows the instantiation for the TMR module 1.

3) Proposed Majority Voter Circuit: The TMR scheme involves two-times duplication of the simplex system hardware, with majority voter ensuring correctness provided at least two out of three copies of the system remains operational. Various majority voter designs have been proposed for the last couple

```
// c17
// Original design
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

// c17
// TMR Module 1
module c17_1 (N1,N2,N3,N6,N7,N22_tmr1,
    N23_tmr1);
input N1,N2,N3,N6,N7;
output N22_tmr1, N23_tmr1;
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_tmr1, N10, N16);
nand NAND2_6 (N23_tmr1, N16, N19);
endmodule
```

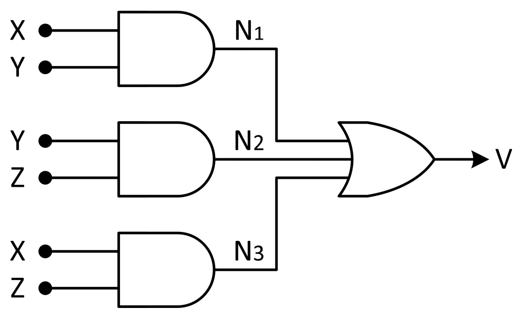
Fig. 4. Code snippet (original design and modified design).

```
// Instantiate the module
c17_1 inst_tmr1 (
.select1(select1),
.N1(N1),
.N2(N2),
.N3(N3),
.N6(N6),
.N7(N7),
.N22_tmr1(N22_tmr1),
.N23_tmr1(N23_tmr1)
);
```

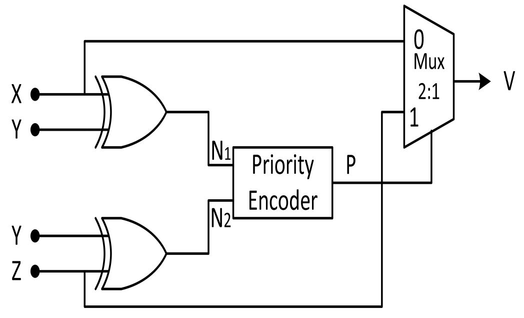
Fig. 5. Code snippet (instantiation).

of years. Fig. 6 shows the schematic diagram of majority voter circuits. Fig. 6 (a) shows the classical voter design. Some other majority voter logics are proposed by Kshirsagar and Patrikar voter circuit [16], Ban and Naviner majority voter [17], and Balasubramanian and Prasad majority voter circuit [7]. Fig. 6 (b-d) show the schematic of other proposed majority voter circuits, respectively.

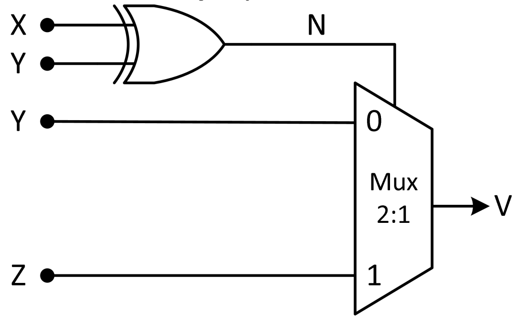
In this work, authors proposed another simple way to represent majority voter logic and added to the RASP-TMR tool as shown in Fig. 7. In this figure, `T1,T2` and `T3` are the inputs of the majority voter circuit. Proposed MVC consists of an `AND, OR` gates and a `multiplexer (2:1)`. Table I shows the functional verification and correctness of the proposed voter logic.



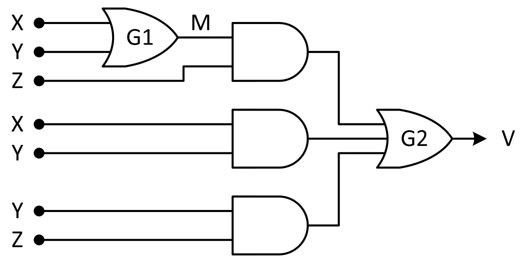
(a) Classical Majority Voter Circuit (MVC).



(b) Kshirsagar and Patrikar MVC [16].



(c) Ban and Naviner MVC [17].



(d) Balasubramanian and Prasad MVC [7].

Fig. 6. Various majority voter designs in the literature.

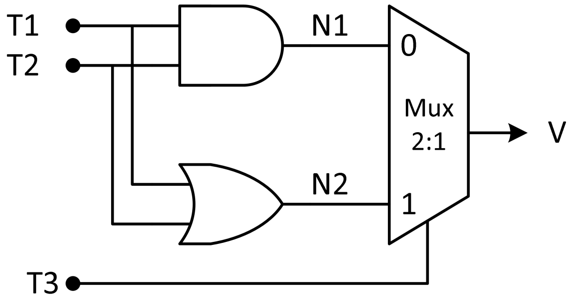


Fig. 7. Proposed majority voter schematic diagram.

TABLE I. TRUTH TABLE VERIFICATION OF PROPOSED MAJORITY VOTER LOGIC FOR TMR

T3	T2	T1	N1	N2	V
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

III. RESULT AND DISCUSSION

The primary purpose of using a TMR design methodology is to remove all single point of failure from the design. If two redundant modules are simultaneously upset, then the output can not be guaranteed to be correct. In order to develop a TMR logic for a target system at a code level, the code needs to be modified for each module and also majority voter circuit is included in the design. This is a non-trivial and time-consuming task.

A. Synthesizable Designs

Keeping these points in mind, the RASP-TMR code generator tool is developed for Verilog HDL designs. This tool not only triplicates the target systems but also generates the top file (named *TMR_TopFile*). The components of this file are already described in Section II in detail. At the graphical user interface, the user needs to provide only a synthesizable Verilog HDL design as an input (This verifies our claim about the tool's simplicity and easy to use). A simple benchmark circuit (c17.v) from ISCAS'85 is considered as an example for illustration of the whole process. A project is developed using Xilinx project navigator. The synthesis of the design is performed and generated the RTL schematic of the example shown in Fig. 8. This proves that the designs generated by RASP-TMR tool are synthesizable.

B. Timing Analysis

This tool is fast and automatically generate the TMR designs. To prove this point, authors have generated TMR designs for various benchmark designs, written in Verilog. In this work, ISCAS85, ISCAS89, and EPFL combinational and sequential benchmark designs are considered for their generation of TMR technique and measured the time. Tables II to V show the size of the designs in terms of a number of logic gates. Last columns of these tables show the time taken by this tool in Seconds. It is noted that the TMR approach for the designs, consisting of thousands of gates, are generated in a fraction of the second by the RASP-TMR tool. The design "hypotenuse" is the largest design among them, which consists

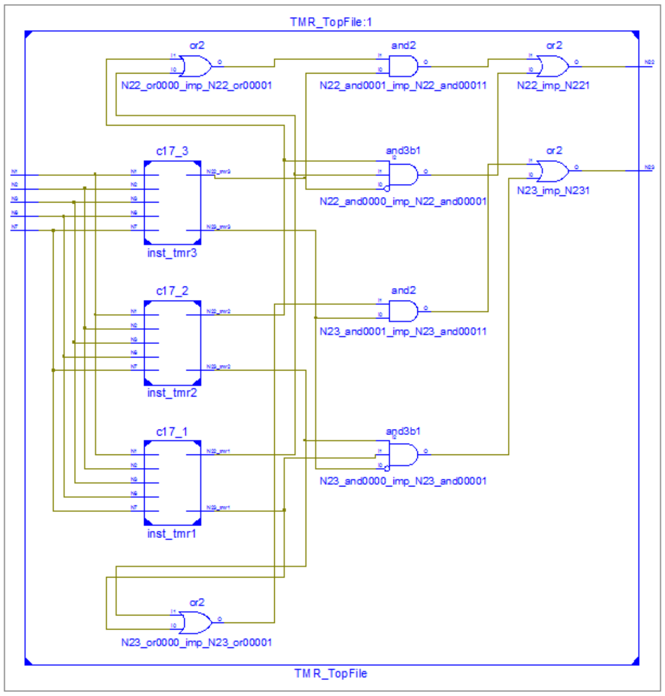

TMR_TopFile:1
or2
N22_or0000_imp_N22_or00001
and2
N22_and0001_imp_N22_and00011
or2
N22_imp_N221
c17_3
inst_tmr3
and3b1
N22_and0000_imp_N22_and00001
or2
N23_imp_N231
c17_2
inst_tmr2
and2
N23_and0001_imp_N23_and00011
c17_1
inst_tmr1
and3b1
N23_and0000_imp_N23_and00001
or2
N23_or0000_imp_N23_or00001
TMR_TopFile

Fig. 8. RTL schematic of c17 circuit with TMR and proposed MVC.

TABLE II. Time Required for ISCAS'85 Combinational Benchmark Designs

S. No.	Benchmark circuits	No. of logic gates	Time (Seconds)
1	c17	6	0.096
2	c432	160	0.1048
3	c499	202	0.2124
4	c880	383	0.1109
5	c1355	546	0.7257
6	c1908	880	0.4743
7	c2670	1269	0.6975
8	c3540	1669	0.3416
9	c5315	2307	0.4948
10	c6288	2416	0.3434
11	c7552	3513	0.5694

of 214335 logic gates and RASP-TMR took approximately 390 seconds.

C. Functional Verification of Proposed Majority Voter Circuit

In this tool, authors also proposed a new majority voter logic. In [7], authors describe the procedure to calculate the Fault Mask Ratio (FMR) for various majority voter circuits. According to them, *FMR is the ratio of a total number of correct voter output states divided by the total number of likely internal and/or external faults corresponding to the applied primary inputs.*

For classical majority voter circuit, the value of FMR is 42.86%. Similarly, for other majority voter circuits Fig. 6 (b-d), the values of FMR are 70.83, 50, 75% respectively. In our case, the calculated FMR for the proposed majority voter circuit is 50% which is better than the classical approach. Authors simulated the proposed majority voter circuit with all possible combinations of inputs. It can be seen from Fig. 9 that the voter signal has a value logic '0' if the majority of inputs are at logic '0' and vice versa. Hence, this simulation verifies the operation of proposed majority voter logic. It is noted that the

TABLE III. Time Required for ISCAS'89 Sequential Benchmark Designs

S. No.	Benchmark circuits	No. of logic gates/FFs	Time (Seconds)
1	s1238	508/18	0.0643
2	s1423	657/74	0.0700
3	s1488	653/6	0.0779
4	s1494	647/6	0.0784
5	s5378	2779/179	0.3016
6	s9234	5597/211	0.5925
7	s13207	7951/638	1.0365
8	s15850	9772/534	1.2942
9	s35932	16065/1728	3.7814
10	s38417	22179/1636	4.0921
11	s38584	19253/1426	5.0263

TABLE IV. Time Required for EPFL (Arithmetic) Benchmark Designs

S. No.	Benchmark circuits	No. of logic gates	Time (Seconds)
1	Adder	1020	0.1985
2	B-shifter	3336	0.4048
3	Divisor	44762	25.265
4	Hypotenuse	214335	390.79
5	Log2	32060	6.8845
6	Max	2865	0.5062
7	Multiplier	27062	5.9782
8	Sine	5412	0.5089
9	Square	24618	5.6014
10	Square-root	18484	3.3004

TABLE V. Time Required for EPFL (Random/Control) Benchmark Designs

S. No.	Benchmark circuits	No. of logic gates	Time (Seconds)
1	Arbieter	11839	1.4639
2	ALU	174	0.0533
3	Coding	693	0.0862
4	Decoder	304	0.1889
5	I2C	1342	0.1995
6	Int-to-float	260	0.0484
7	Memory	46836	32.9611
8	Encoder	978	0.1091
9	Router	257	0.0754
10	Voter	13758	2.1622

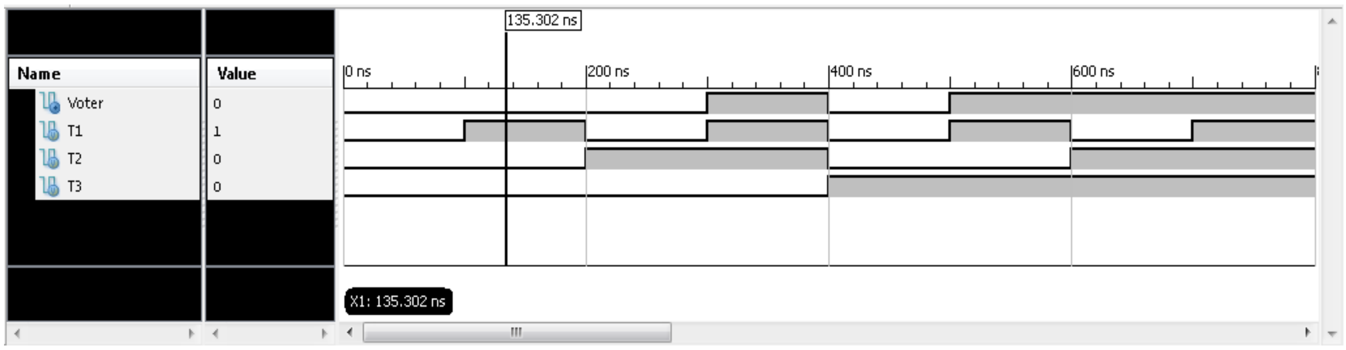


Fig. 9. Simulation results for the validation of proposed majority voter circuit.

shaded area of signals T1, T2, T3, and Voter represent the logic ‘1’ value.

D. Simulation Verification using Fault Injection Technique

Fault injection is a useful technique to test the integrity of the TMR system. In order to test and verify the fault tolerance capability of the target design developed under the RASP-TMR tool, simulation of the design is carried-out with the help of Xilinx ISE Design Suite 13.4 tools and ISim simulator. Xilinx ISE is used to develop the project and writing a test bench, while the simulation is performed by ISim tool. A simulation environmental set-up is created and Fig. 10 shows the block diagram of the set-up. It consists of TMR design and a golden (fault-free) module of the design. Faults are injected in the TMR design in each module. When the faults are activated one by one in each module and different combinations of input are applied and then responses are compared. If the comparator output is logic ‘0’, that means both the golden and TMR outputs are the same and the TMR approach masks the faulty module perfectly. In order to validate the approach, a simple benchmark design from ISCAS’85 combinational circuits has chosen. Authors have injected a total of 12 faults in the c17 benchmark circuit in different locations of the design, 4 faults in each TMR module. The way of injecting faults in the Verilog design code is described in our previous work [3], [18], [19].

In the first instance, the ‘fault 0’ to ‘fault 3’ are activated in module 1 of the TMR design using the 2-bit vector `faultIn1`. The input patterns are applied and denoted by input ports `(N1, N2, N3, N6, N7)`. The ‘cmp1’ signal compares the responses of the golden module with the responses of the TMR approach. Fig. 11 shows that the signal ‘cmp1’ is continuously at logic ‘0’, which means both the responses are the same. Similarly, the same approach is repeated for the second and third module of the TMR approach and simulation results are shown in Fig. 12 and 13. Hence, it verifies the operations of the TMR approach developed by the RASP-TMR tool and the proposed majority voter circuit.

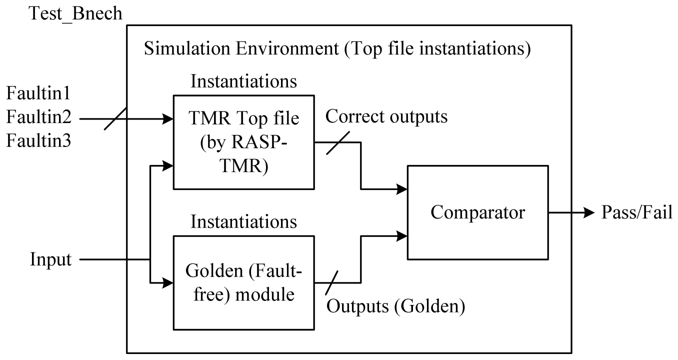


Fig. 10. Simulation environment for the verification of proposed tool.

IV. Conclusion

In this work, authors developed a RASP-TMR tool which triplicates any combinational and sequential digital designs. The RASP-TMR tool is simple, fast and user-friendly. The user needs to provide a synthesizable Verilog file as an input, and then the tool creates TMR design along with the proposed majority voter logic circuit. TMR design for various combinational and sequential benchmark circuits is generated and evaluated. The results show that the RASP-TMR tool takes less time to generate the synthesizable Verilog code with TMR technique. Benchmark designs are simulated using Xilinx tools to evaluate the features of our proposed tool.

The future work will add more features to the tool such as TMR with Multiple Voters, Duplication With Comparison (DWC) and N-modularity redundancy for the evaluation of the fault-tolerant capability of FPGA-based designs conveniently.

References

[1] G. Corradi, R. Girardey, and J. Becker, “Xilinx tools facilitate development of FPGA applications for IEC61508,” in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, Erlangen, Germany, Jun 2012, pp. 54–61.

[2] 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, Dec 2014, pp. 68–74.

[3] A. R. Khatri, A. Hayek, and J. Börcsök, *Applied Reconfigurable Computing*, ser. Lecture Notes in Computer Science, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Cham: Springer International Publishing, 2016, vol. 9625. [Online]. Available: http://link.springer.com/10.1007/978-3-319-30481-6

[4] M. Desogus, L. Sterpone, and D. M. Codinachs, “Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs,” in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. Platja d’Aro, Girona, Spain: IEEE, Jul 2014, pp. 111–115. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6873681

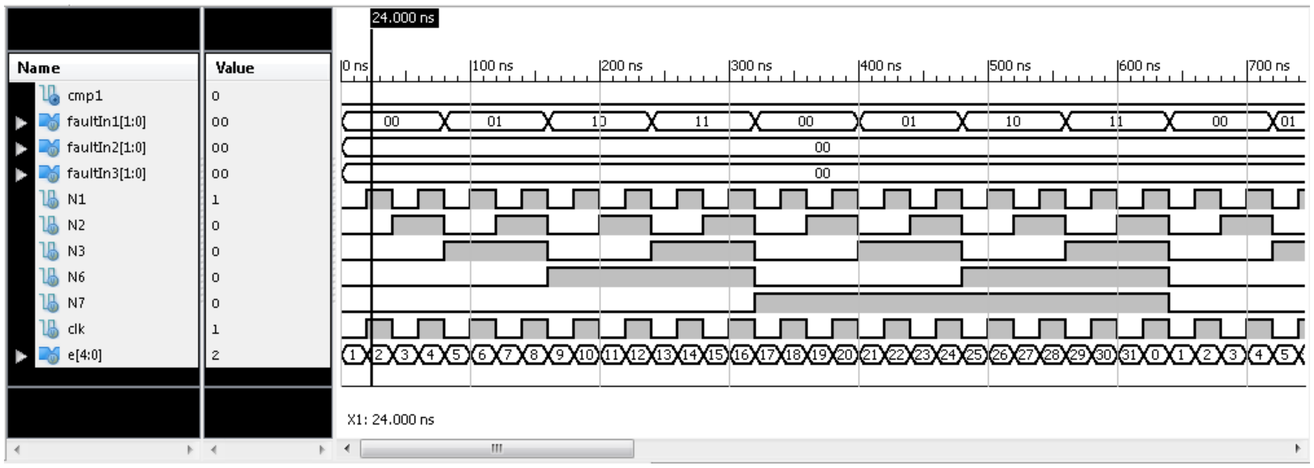


Fig. 11. Simulation results for bit-flip injection in TMR module 1.

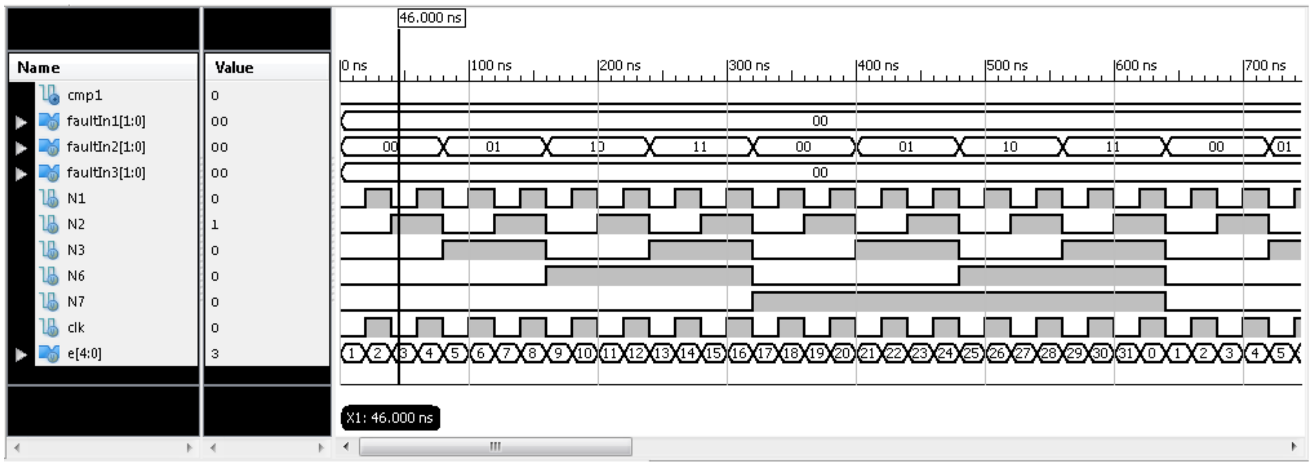


Fig. 12. Simulation results for bit-flip injection in TMR module 2.

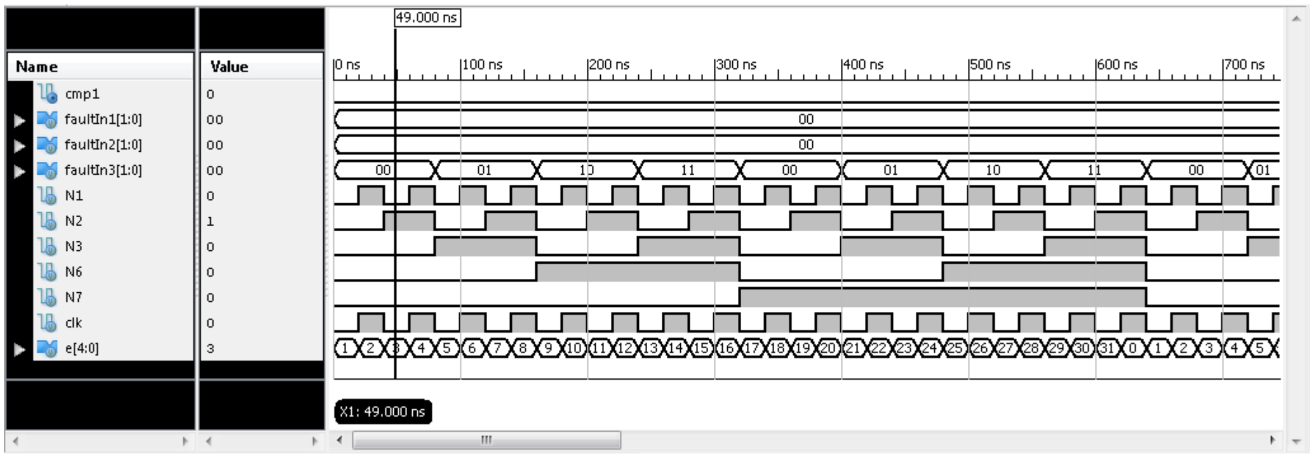


Fig. 13. Simulation results for bit-flip injection in TMR module 3.

[5] C. Frenkel, J.-d. Legat, and D. Bol, "A Partial Reconfiguration-based scheme to mitigate Multiple-Bit Upsets for FPGAs in low-cost space applications," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. Bremen: IEEE, Jun 2015, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7238095

[6] W. Xin, "Partitioning Triple Modular Redundancy for Single Event Upset Mitigation in FPGA," in *2010 International Conference on E-Product E-Service and E-Entertainment*. Henan: IEEE, Nov 2010, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5660842

[7] P. Balasubramanian, K. Prasad, and N. E. Mastorakis, "A Fault Tolerance Improved Majority Voter for TMR System Architectures," *WSEAS Transactions on Circuits and Systems*, vol. 15, pp. 108–122,

2016. [Online]. Available: http://arxiv.org/abs/1605.03771

[8] S. Müller, T. Koal, M. Schölzel, and H. T. Vierhaus, "Timing for Virtual TMR in Logic Circuits," in *IEEE 20th InternationalOn-Line Testing Symposium (IOLTS)*, 2014, pp. 190–193.

[9] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and A. Vallero, "A novel methodology to increase fault tolerance in autonomous FPGA-based systems," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. Girona, Spain: IEEE, Jul 2014, pp. 87–92. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6873677

[10] P. Samudrala, J. Ramos, and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, Oct 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1344451

[11] Xilinx, "Xilinx TMRTool User Guide - TMRTool Software Version 13.2," Tech. Rep., 2017. [Online]. Available: https://www.xilinx.com/support/documentation/user{_}guides/ug156-tmrtool.pdf

[12] S. Kulis, "Single Event Effects mitigation with TMRG tool," *Journal of Instrumentation*, vol. 12, no. 01, pp. C01 082–C01 082, Jan 2017. [Online]. Available: http://stacks.iop.org/1748-0221/12/i=01/a=C01082?key=crossref.0735729cdfa9a0dabd055faa790f2459

[13] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*. IEEE, 2006, pp. 226–232. [Online]. Available: http://ieeexplore.ieee.org/document/4017162/

[14] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, no. 3. IEEE, Apr 2017, pp. 129–132. [Online]. Available: http://ieeexplore.ieee.org/document/7966665/

[15] L. A. C. Benites and F. L. Kastensmidt, "Automated design flow for applying Triple Modular Redundancy (TMR) in complex digital circuits," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*. IEEE, Mar 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8349668/

[16] R. Kshirsagar and R. Patrikar, "Design of a novel fault-tolerant voter circuit for TMR implementation to improve reliability in digital circuits," *Microelectronics Reliability*, vol. 49, no. 12, pp. 1573–1577, Dec 2009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0026271409003114

[17] T. Ban and L. A. Naviner, "A simple fault-tolerant digital voter circuit in TMR nanoarchitectures," in *Proceedings of the 8th IEEE International NEWCAS Conference 2010*. IEEE, Jun 2010, pp. 269–272. [Online]. Available: http://ieeexplore.ieee.org/document/5603933/

[18] A. R. Khatri, A. Hayek, and J. Börcsök, "ATPG method with a hybrid compaction technique for combinational digital systems," in *2016 SAI Computing Conference (SAI)*. London, UK: IEEE, Jul 2016, pp. 924–930. [Online]. Available: http://ieeexplore.ieee.org/document/7556091/

[19] A. R. Khatri, A. Hayek, and J. Börcsök, "Validation of the Proposed Fault Injection , Test and Hardness Analysis for Combinational Data-flow Verilog HDL Designs under the RASP-FIT Tool," in *IEEE 16th Int. Conf. on Dependable, Autonomic & Secure Comp., 16th Int. Conf. on Pervasive Intelligence & Comp., 4th Int. Conf. on Big Data Intelligence & Comp., and 3rd Cyber Sci. & Tech. Cong.* Athens, Greece: IEEE Comput. Soc, Aug 2018, pp. 544–551.