Single Event Effects mitigation with TMRG tool

Published by IOP Publishing for Sissa Medialab

Received: *November 11, 2016*
Accepted: *January 26, 2017*
Published: *January 26, 2017*

Topical Workshop on Electronics for Particle Physics,
26–30 September 2016,
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Single Event Effects mitigation with TMRG tool

S. Kulis

CERN,
Geneva, Switzerland

E-mail: szymon.kulis@cern.ch

Abstract: Single Event Effects (SEE) are a major concern for integrated circuits exposed to radiation. There have been several techniques proposed to protect circuits against radiation-induced upsets. Among the others, the Triple Modular Redundancy (TMR) technique is one of the most popular. The purpose of the Triple Modular Redundancy Generator (TMRG) tool is to automatize the process of triplicating digital circuits freeing the designer from introducing the TMR code manually at the implementation stage. It helps to ensure that triplicated logic is maintained through the design process. Finally, the tool streamlines the process of introducing SEE in gate level simulations for final verification.

Keywords: Digital electronic circuits; Front-end electronics for detector readout; Radiation-hard electronics; VLSI circuits

doi:10.1088/1748-0221/12/01/C01082

Contents

1 Introduction

Single Event Effects (SEE) are a major concern for integrated circuits used in a radiation environment, especially for circuits fabricated in modern deep sub-micron technologies. For reliable system operation in environments such as the Large Hadron Collider (LHC) it is necessary to protect the logic from radiation-induced SEE. Many techniques have been proposed in order to protect the circuit against SEE. Virtually all methods rely on data redundancy. It is assumed that if the information is stored in several places (circuit nodes), it can be properly reconstructed even if some of these nodes are disturbed. Among the SEE hardening techniques, some are based on hardening standard cells [1, 2] while others address the problem on a system level, by utilizing error correcting coding (ECC) [3], temporal redundancy [4], or Triple Module Redundancy (TMR) [5, 6].

The Triple Module Redundancy Generator (TMRG) [7] tool developed at CERN automatizes the process of triplicating digital circuits freeing the designer from introducing the TMR code at the implementation stage. The TMRG tool is part of the toolset assisting the user during the whole digital implementation flow. In this paper, the TMRG toolset is presented. A typical digital design flow incorporating TMRG toolset is presented in section 2, followed by examples and discussion of the TMRG tool architecture in section 3 and 4 respectively.

2 Digital design flow with TMRG toolset

The TMRG toolset is integrated with the digital flow as shown in figure 1. The main components of toolset are: the TMRG, the Placement Generator (PLAG), the Single Event Effect Generator (SEEG), and the Test Bench Generator (TBG).

As detailed in figure 1, one can see that TMRG adds several new steps with respect to the standard digital design flow and affects some other steps. The philosophy of the tool is that a user writes regular (non-triplicated) Verilog code and then uses the TMRG tool to transform it to code containing triplicated elements. Automatizing this step reduces the coding time and minimizes the probability of introducing errors during the manual triplication process. The behavior of the tool can be controlled by user-applied constraints (described in more details in section 3).

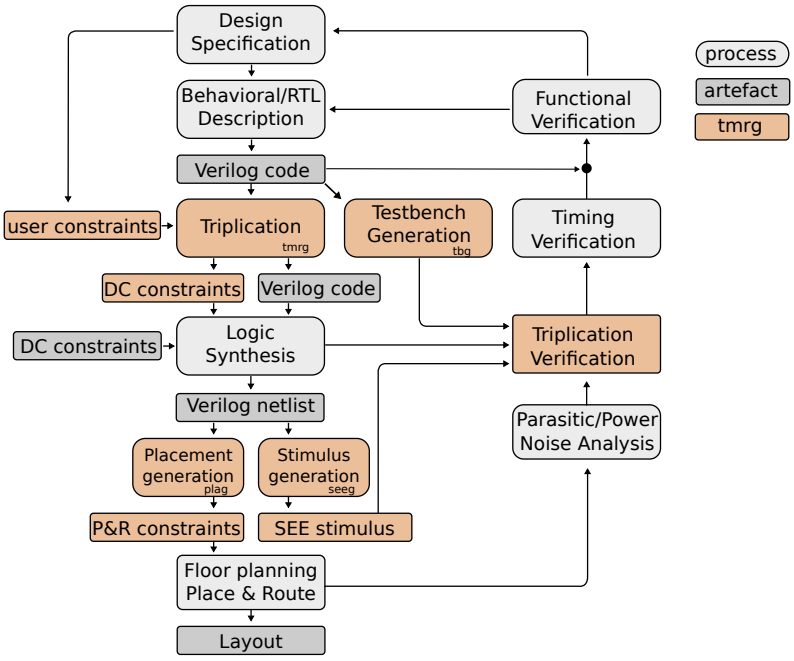

Figure 1. Digital design flow including TMRG components: Triple Module Redundancy Generator (TMRG), Placement Generator tool (PLAG), Single Event Effect Generator (SEEG), Test Bench Generator (TBG).

The code generated by TMRG tool will contain redundancy and therefore the synthesizer will want to remove it, compromising the production of SEU/SET robust digital circuits. The TMRG tool helps to ensure that triplicated logic is maintained during the synthesis process by generating a set of constraints for the synthesis process.

In modern deep sub-micron technologies, the probability of multiple bit upsets caused by the same particle is not negligible. In order not to compromise TMR structures, the redundant cells should be placed away from each other so that one particle cannot affect multiple cells storing the same redundant information. The PLAG tool was created to streamline the place and route process by generating constraints which ensure that triplicated cells are not placed too close to each other.

The TMRG toolset is completed by two tools assisting the user in the verification process. The first one is SEEG which provides a uniform mechanism to introduce in a random manner SEU and SET in a gate level transient simulation. The second tool TBG creates a test bench instantiating the Device Under Test (DUT) in non-triplicated and triplicated versions, the required fan-outs and voters, as well as the SEE generator.

3 Constraining the design

In order to ensure maximum robustness against SEU and SET one should triplicate all circuitry in the chip. Unfortunately, it is not always possible as the triplication does come with penalties. Just to name a few, one should mention an increase in occupied area, an increase in power consumption, and a reduction in the maximum clock frequency. Moreover, not all blocks can be easily triplicated, e.g. I/O ports or some analog blocks. The TMRG tool allows the designer to decide which blocks

2017 JINST 12 C01082

and signals should be triplicated. It also automates the conversion process between triplicated and non-triplicated signals. There are two basic conversion schemes: if a non-triplicated signal is connected to a triplicated signal a simple passive fan-out is added, if a triplicated signal is connected to a non-triplicated signal a majority voter is added. There are several ways to pass constraints to the TMRG tool, the simplest and the most intuitive is to put directives directly in the source code. The directives are placed in comments, so they do not affect other tools. For the sake of the demonstration, let's consider only three directives:

```
// tmrg default [triplicate|do_not_triplicate]
// tmrg triplicate netName
// tmrg do_not_triplicate netName
```

The first directive specifies the default behavior for the entire module. The default behavior can be changed for individual nets and registers using directives from lines 2 and 3. Let us consider a simple combinatorial module:

```
module comb (in,out);
  input in;
  output out;
  wire combLogic;
  assign combLogic = ~in;
  assign out = combLogic;
endmodule
```

which is schematically presented in the center of figure 2. By applying different constrains the designer has control over how the TMRG tool triplicates the design. Applying the `default triplicate` constraint leads to the triplication of all elements (see figure 2a). If `do_not_triplicate` directive is used for a signal, a fan-out or majority voter is inserted respectively as shown in figures 2b and 2c. The TMRG tool does not assume any particular architectures, therefore the designer may generate any variation of the circuit as presented in figures 2d and 2e.

Using constraints presented above one can triplicate any circuit, including a state machine. Among others one can obtain the three most popular types of triplication: a) triplication of registers only, b) triplication of registers and clock signals, c) triplication of registers, combinatorial logic, voters, and clocks. At this point it should be mentioned, that because of the tool's flexibility it is possible to generate designs which despite being triplicated are not immune to SEE.

The constraints presented above are only a subset of all available constraints supported by the TMRG tool. Among the others, the TMRG features constraints allowing for accessing voter error output (to be used for error detection), accessing individual signals from a triplicated bus, and specifying various voting and fan-out cells. Moreover, TMRG has features enabling the user to instantiate already triplicated code, while still being able to benefit from other components of the toolset.

4 Architecture and implementation

The TMRG toolset is implemented in Python programming language [8]. Python offers cross-platform runtime environment rich in standard libraries. The slower (compared to compiled languages) execution time is offset by the fast development time.

2017 JINST 12 C01082

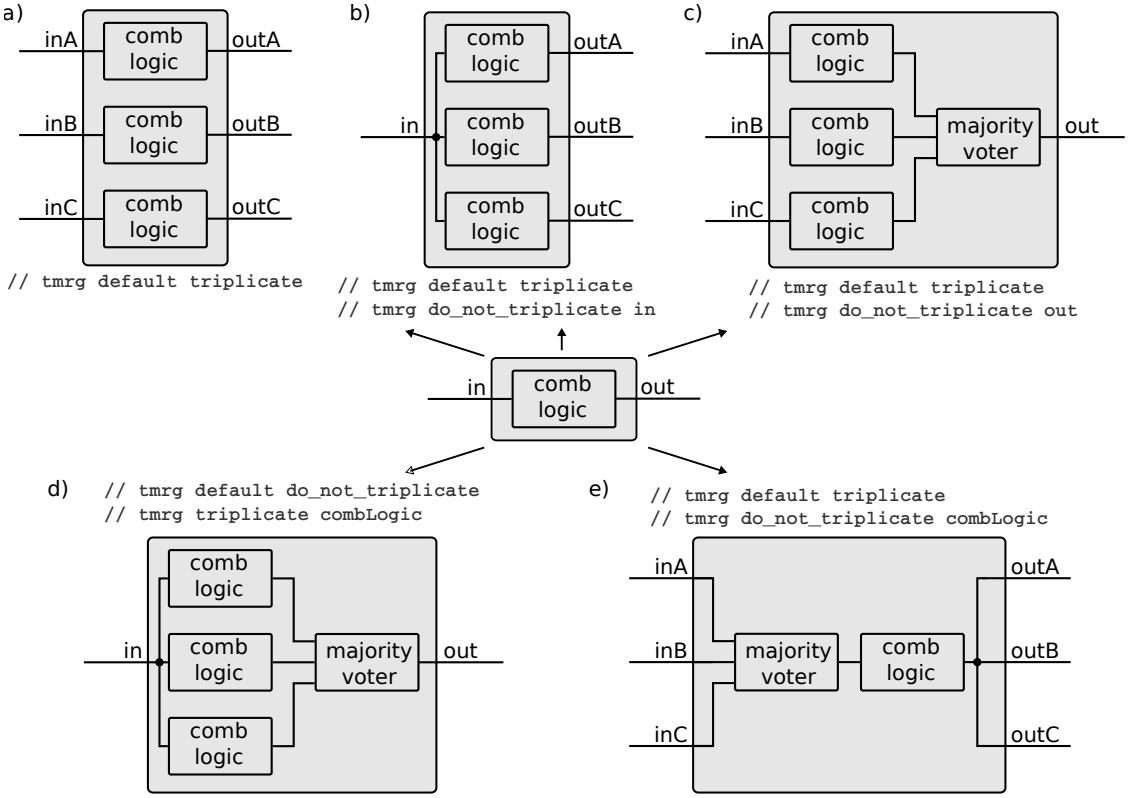

Figure 2. An impact of TMRG constraints on the generated code. The same code (represented by the circuit in the center of diagram) leads to different outputs depending on applied constraints.

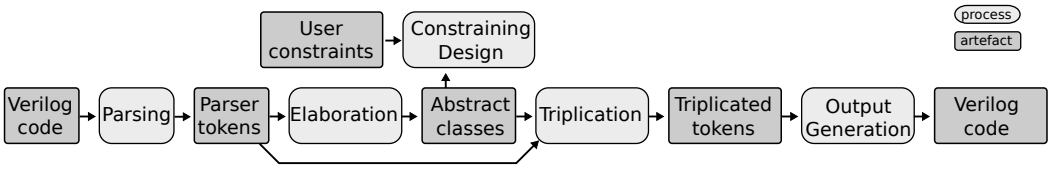

Figure 3. Architecture of TMRG tool.

The TMRG flow is presented in figure 3. In the first phase, files are loaded and parsed by parser based on Pyparsing library [9]. Tokens obtained from the parsing process are elaborated. In this step, the structure of the code is analyzed, and essential information (like signal names, directions of ports) are captured. Afterwards, the module hierarchy for the whole project is built. In the next step, user constraints are applied to the design. The constraints may be passed to the application in various ways, including comments placed in the source code, command line, or an external configuration file. It is important to mention that the constraints propagate automatically through the design hierarchy. In the next step, the triplication process itself takes place. The triplication process uses tokens and user constraints as inputs and produces triplicated tokens and Design Compiler constraints at its output. Finally, the output code is generated from the triplicated tokens. Using this approach, it is possible to ensure that the output code is human readable and formatted in a consistent style across all output files.

5 Summary and outlook

In this paper, the TMRG toolset has been presented. The TMRG toolset assists the user along the process of designing electronics resistant to Single Event Effects. The TMRG tool targets the typical ASIC design flow commonly used in the HEP community, however, it can be also used to triplicate Verilog code targeting FPGA applications. At the moment, the TMRG tool is not able to generate specific constraints for the FPGA synthesis tools, but, due to the tool modular architecture, such support can be added as a plug-in. Due to its architecture, it only lightly constrains the user's coding style while allowing to obtain various flavors of TMR (registers only, registers and clock, full triplication). Besides triplicating the user code, the toolset assists the user in the physical implementation stage (synthesis, P&R) and the verification stage (generation of SEE). The TMRG toolset was already used in several projects, reducing their development time.

Acknowledgments

The author would like to acknowledge colleges from CERN microelectronic group for numerous discussions about the SEU mitigation techniques and feedback about the TMRG tool. Many thanks to Paulo Moreira, Pedro Vicente Leitao, Xavi Llopart Cudie, Tuomas Poikela, Cesar Marin Tobon, and Jorgen Christiansen.

References

[1] S. Whitaker, J. Canaris and K. Liu, *SEU hardened memory cells for a CCSDS Reed-Solomon encoder*, *IEEE Trans. Nucl. Sci.* **38** (1991) 1471.

[2] F. Faccio et al., *Single event effects in static and dynamic registers in a 0.25 μm CMOS technology*, *IEEE Trans. Nucl. Sci.* **46** (1999) 1434.

[3] T.K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, Wiley, 22 July 2005.

[4] W. Wang and H. Gong, *Edge triggered pulse latch design with delayed latching edge for radiation hardened application*, *IEEE Trans. Nucl. Sci.* **51** (2004) 3626.

[5] J. Von Neumann, *Probabilistic Logics*, Auromafa Studies, Princeton University Press (1956).

[6] E.F. Moore and C. Shannon, *Reliable Circuits Using Less Reliable Relays. Part I*, *J. Franklin Inst.* **262** (1956) 191; *Reliable Circuits Using Less Reliable Relays. Part II*, *J. Franklin Inst.* **262** (1956) 281.

[7] TMRG tool, https://cern.ch/tmrg.

[8] Python programming language, https://www.python.org/.

[9] P. McGuire, *Getting Started with Pyparsing*, O'Reilly Media (2007).