2013 16th Euromicro Conference on Digital System Design

Register Transfer Level Workflow for Application and Evaluation of Soft Error Mitigation Techniques

Filipe Sousa
CERN and Faculty of Engineering
University of Porto, Portugal
filipe.sousa@cern.ch

Francis Anghinolfi
CERN
Switzerland

João Canas Ferreira
INESC TEC and Faculty of Engineering
University of Porto, Portugal
jcf@fe.up.pt

***Abstract*—Digital circuits exposed to environments with high levels of radiation, such as those found in High Energy Physics experiments, are prone to Single Event Upsets. These upsets impact the reliability of the circuit. In order to mitigate the effects of the upsets, several well-known techniques for use with register transfer level (RTL) circuit descriptions have been proposed over the years. They typically have a large impact on circuit size and power consumption. Therefore, they are often applied only to the more critical modules of the system. Additionally, the manual implementation of those techniques has a significant cost in terms of time and design effort, involving both RTL changes and tailoring of the synthesis flow to avoid optimizing away the additional hardware. This paper describes an automated workflow that reduces the time for implementing SEU mitigation techniques, avoids the errors caused by manual alteration of the RTL descriptions, and enables the designer to explore different alternatives quickly. The paper describes the application of the workflow to three digital circuits and discusses the data obtained from the implementation of the different mitigation techniques.**

I. Introduction

Digital integrated circuits fabricated in sub-micron technologies employ lower supply voltages than the ones used with older technologies. In addition, circuit nodes have smaller capacitance, and hence less critical charge. These two factors lead to the increase of the number of soft errors (Single Event Upsets) that occur due to radiation. Single Event Upsets (SEUs) are generated when a charged particle hits the silicon and changes the logic state of a single node [1]. For high energy physics experiments, where the radiation level is expected to be high the new generation of detectors and its electronics must address the challenge of delivering circuits capable of operating under high radiation levels without increasing the overall power requirements.

In order to circumvent or mitigate SEU effects many approaches have been described in the literature. One possibility is to change the configuration of the storage cells (flip-flops) by adding redundant paths [2], [3], [4] or by using ratioed transistors with specific sizes and strength [5]. These approaches are useful if the corresponding libraries are already developed and available, but otherwise they are time consuming due to the time spent on the design and characterization of such cells. These approaches also have the limitation of being technology-dependent: the cells need to be redesigned every time a new technology node is used.

Mitigation techniques that can be applied at the register transfer level (RTL) are generally technology independent. Therefore the same technique can be used for different technology nodes without any adaptation. Triple Modular Redundancy (TMR) [6], [7], [8] and the use of Error Detection and Correction (EDAC) codes, e.g. Hamming codes [9], are two examples of techniques that can be implemented in RTL.

When working on RTL descriptions, the designer is mainly concerned with implementing the desired functionality and, therefore, should stay focused on that goal. The addition of TMR to an already designed circuit can be difficult due to the number of signals that need to be triplicated and connected. The introduction of TMR, if made manually, is also time consuming, and error prone. And if the errors can ultimately be detected in a later stage they just add up to the time needed to finish a design. The same observation applies to the introduction of EDAC techniques to an already designed circuit.

Applying a single technique to increase SEU robustness to the entire design is not a very efficient method, since the design has several modules, and each has a different organization and requirements. In order to study the SEU susceptibility of the different modules in a design, a simulation tool capable of inserting bit-flips in a given node can be used [10], [7].

This paper introduces a workflow for systematically adding TMR and Hamming codes to an RTL design, and for evaluating its SEU robustness. The objective of the work is to validate a flow for introducing protective measures against SEUs in a semi-autonomous way, in order to reduced the time that a designer spends while protecting a digital circuit. For this work flow, a tool capable of inserting bit flips in a digital simulation was developed, in order to provide the designer with information about individual module's robustness to SEU events. Specifically, the contributions of this work are:

- A complete workflow for systematically enhancing RTL designs;
- Automatic application of SEU effects mitigation techniques to existing RTL designs;
- An automatic fault injection procedure integrated in a standard simulation environment, as a tool to support SEU robustness evaluation.

Additionally, the paper reports data about the impact of different techniques on power consumption for three benchmark circuits synthesized for a 130 µm CMOS technology.

The work here develop is to be applied in tracker system of the ATLAS (A Toroidal LHC ApparatuS) HEP experiment

`/13 $26.00 © 2013 IEEE
DOI 10.1109/DSD.2013.92

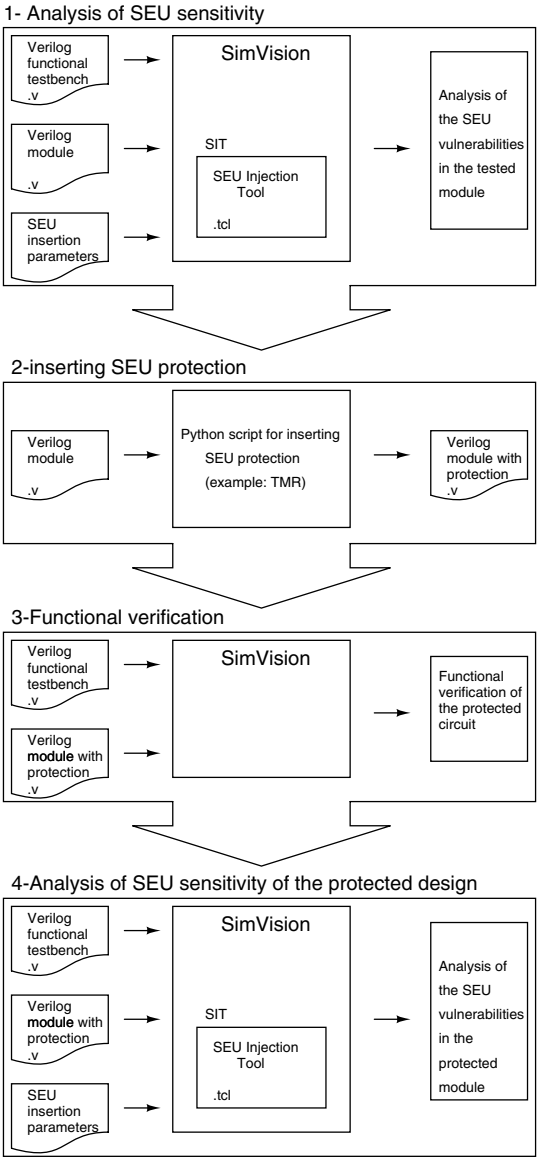

Fig. 1. Flowchart describing the supported workflow.

in the Large Hadron Collider (LHC) at CERN .

In the following section the flow for implementing SEU protection is presented. In Sec. III the techniques for adding robustness to the circuits are described. Section IV presents a tool for inserting bit-flips in a digital circuit. Section V describe the implementation of the flow presented in this paper, and the results obtained are shown. Finally, Sec. VI presents the conclusions.

II. Workflow for Implementing SEU Protection

This section describes the workflow for adding SEU protection to a already existing RTL module written in Verilog. A representation of the flow can be seen in Fig. 1. The current implementation is integrated in the commercial SimVision graphical debugging environment.

The designer must have a working Verilog testbench, that is used for functional validation of the design. The target Verilog module must run in the provided testbench without error. The designer must also define a set of parameters for introducing SEU like bit-flips during the simulation. These parameters are the number of SEUs that will be inserted, the minimum period between two adjacent SEUs (important to avoid multiple bit upsets), when they should be created (either randomly or in moments specified by the designer), and in which nodes they can be created.

In order to perform an evaluation of the SEU vulnerabilities of the design, the *SEU injection tool* (SIT) is run. This tool, written in TCL [11], reads the parameter file, and executes two simulations. One simulation without SEUs that will be used as a golden simulation, and a second simulation where the SEUs are generated accordingly to the defined parameters. The designer can check the output of the testbench for discrepancies or he can use the SimVision compare tool to check the differences between the golden simulation and the SEU simulation. If after an SEU the circuit is not capable of returning to the same value that was registered in the golden simulation, it means that the affected nodes are not protected against SEUs. At this point the designer should take a decision of which parts of the design should be protected and which parts can be left unprotected.

In the second step of the flow the original design is protected with a given technique, for example, TMR. The technique is applied by a program written in Python. The output will be another Verilog module with the same functionality of the original but with added SEU protection.

In the third step, the functional validation of the protected design is done by repeating the same simulations of step one. Since the interfaces, input and output signals, of the module are still the same as before, the initial testbench can be used together with an instance of the new protected design instead of the original one. This step confirms that the introduction of the chosen mitigation technique did not affect the module's functionality.

The fourth and the last step is similar to the first step of this flow. The protected design will now be simulated by using the initial testbench and the SIT. This step is crucial for the designer to verify the correct implementation of the protective technique applied in step two. In this step the designer should use the same parameters as before in order to verify that what he has protected is now actually protected.

SEU protection techniques can be applied at different levels of a design. One of the approaches is to apply the technique to the complete design. In this way the entire design is protected against SEU. However, the overhead in power and area will be very high when compared with other approaches.

In complex designs the reliability requirements are not the same for all the parts of a design, so addressing the SEU protection with a global solution may lead to an inefficient usage of area, and power. A more efficient way of implementing protection in a circuit is to apply it in a localized manner, that is, applying specific techniques at the module level. This allows the designer to choose which modules to protect and which techniques to use on them. In this manner a block that does not have a requirement for SEU protection can be left untouched by the protective techniques. By taking into account that the

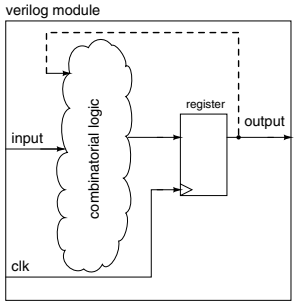

Fig. 2. Representation of a digital circuit

individual modules are designed with different structures (state machines, combinational logic, register banks) a differentiated approach for applying protection against SEUs can be used in order to achieve an even better use of the resources (mainly, power and area). This is the approach supported by the workflow described here. In terms of the Verilog description, the designer can specify different strategies for different modules and for different processes (`always` blocks) in a given module.

The introduction of protective techniques against SEUs in designs implies in most of the cases the systematic rewriting of the RTL code. The introduction of such techniques should be relatively fast in order to not increase the development time. The manual rewriting of code by a designer can be a slow process and, more importantly, it is prone to the introduction of errors, either by missing some signals or by not implementing the techniques correctly. Therefore, it is important to provide a program or tool that performs the task automatically and as transparently as possible.

III. Supported Techniques for SEU Tolerance

For this work two main techniques were implemented in order to protect the digital logic against the effects of an SEU. They were chosen as practical examples of how a circuit can be protected in an advanced state of the design. The first technique is triple modular redundancy, and it uses redundant logic in order to filter out an SEU. The second technique uses parity bits to protect the data that is stored in the registers. A detailed explanation of the implementation of both techniques follows next.

A. Hardware Triplication

Triple Modular Redundancy is used to implement SEUs robustness in the design by triplicating the registers that store the bits from one clock cycle to the next. Triplication is achieved by creating three identical instances; the outputs of those instances are then sent to a majority voter block that will have as output the value that occurs in more than one signal (three if all instances are working properly or two if one had an SEU). Since the registers are inside the instances they need to be loaded with the result from the voter. This implies that the register signals that were previously just used internally in the module (dashed line in Fig. 2) also have to be voted and only this voted value must be used as an input for further computations.

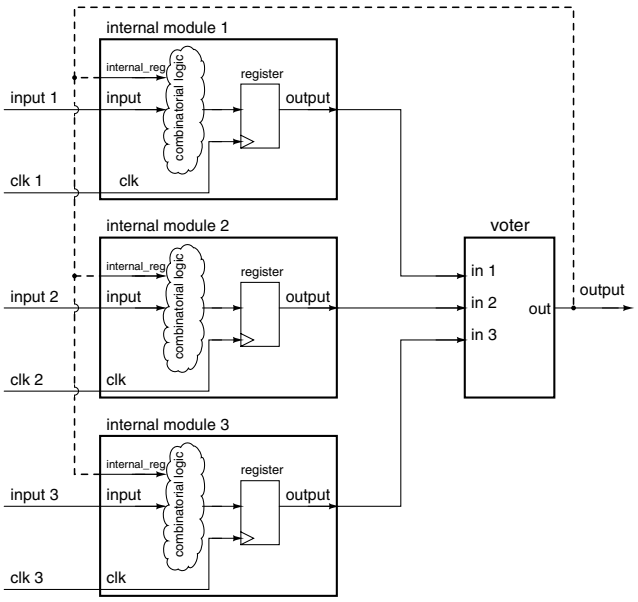

Fig. 3. Triple Modular Redundancy applied to a Verilog module with combinational and sequential logic.

A digital circuit, as the one seen in Fig. 2, can be divided in two domains, the combinational and the sequential logic domains. An SEU can happen due to a transient upset occurring in the combinational logic and propagating to the sequential logic, which registers the wrong value. However, only a fraction of the signals of a combinational logic block are observed at the output at any given moment, thus the probability of an SEU occurring due upsets in the combinational logic is low but not zero. The sequential domain contains the memory elements that store the circuit's state, and that will then be used together with the present input value to compute the outputs of the combinational domain. The memory elements are the most critical part regarding SEU sensitivity, because the stored value is kept during a full clock period without an external driver. Even if the impact of the combinational logic on the SEU robustness is reduced compared to the impact of the sequential part, it should not be neglected. Therefore the triplication scheme should apply triplication not only to the sequential logic, but also to the combinational logic that drives the signals that are to be stored in registers. Due to this factor the triplication that was implemented in this work was applied to both parts of the design. The concept of this triplication scheme can be seen in Fig. 3. In this figure three internal modules are instantiated; these modules contain logic similar to the original module. However all register outputs are routed as outputs of the modules. These outputs are then connected to the (majority) voter through the inputs `in1` to `in3`. From the voter all the signals are routed back (dashed lines) to the three internal modules in order for them to load the correct value into theirs registers. Only the signals that were already original outputs are routed to the outside of the triplicated circuit. The inputs to the triplicated circuit can be same signal for all the three internal modules or, if they already exist independently outside of the circuit, each input can be individual (as in the case of the Fig. 3)

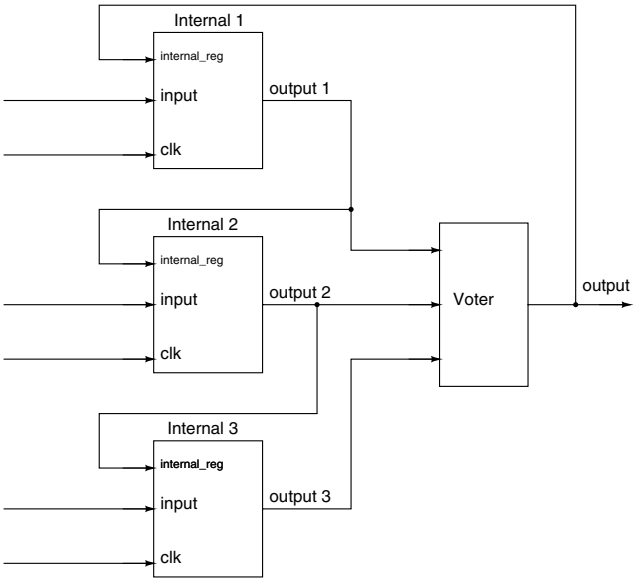

Fig. 4. Chain Triple Modular Redundancy

1) Simple Triplication : The first way of connecting the signals from the output of the voter to the input of the triplicated modules is to connect the voted signals to the inputs of all three internal instances. In this way, the three instances will have the same inputs at all time; therefore, if an SEU occurs in one of the blocks it will be corrected in the following clock cycle as the signal used in the next clock cycle is the voted, corrected, signal. By using this connection scheme the effect of an SEU will be corrected by design in one clock cycle and will not propagate to other blocks. However, there is still one critical node that can corrupt the design in the case of an SEU. The critical node is output from the voting block which connects to the triplicated instances. This node is small in comparision to the size of the internal triplicated instances and the voting block itself, and therefore the probability of occurrence of an SEU on that node is small.

2) Chain Triplication: The other connection method implemented avoids this single node problem by having the output of the voting circuit connected to only one of the triplicated instances. The other two triplicated instances have their inputs connected to the non-voted output of the other triplicated instance. Figure 4 shows the connection scheme. In this scheme instance one has its inputs connected to the voter, instance two has the inputs connected to the output of the instance one, and finally instance three has the inputs connected to the outputs of instance two. With this setup an SEU on the output signal of the voter will not propagate to all three instances and corrupt the following output of the design, as the other two instances will have the correct value. However, instance number one will have its output wrong and this wrong value will propagate in the next clock cycle to the instance two, that is fed directly from the output of instance one. Even in this case the output of the design is still correct, as instance one and instance three have now the correct value, with the error being present only in instance two. The error will propagate in the same way to instance three on the next clock cycle, but will then be definitively eliminated from the system. In this scheme the circuit is robust even in the presence of an SEU in the output of the voting block. The drawback is that the circuit becomes sensitive to another SEU occurring in the three clock cycles after the occurrence of the first SEU at the output of the voting block.

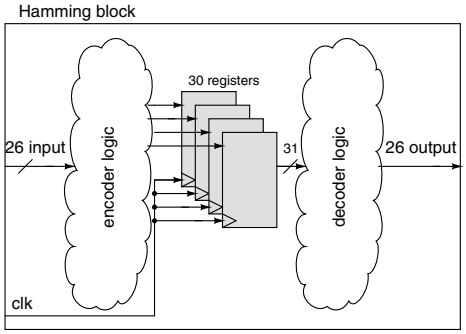

Fig. 5. Hamming block diagram

B. Hamming Encoding

Instead of applying triplication to a design, encoding the data to be stored can be used to reduce the area overhead of the SEU protection. This technique has the disadvantage of not protecting the circuit against those Single Event Transients (SETs) that occur in the combinational logic and that are then latched in the sequential logic, thus creating an SEU.

The implementation of this encoding scheme is composed of three distinct blocks as shown in Fig. 5. The first block, the encoder, has as input the data to be stored in a register. This encoding is done by applying a Hamming encoding where parity bits are added in order to detect and correct a single bit flip in the data. The second block is a regular register, but it is widened to include the parity bits. The third block is a decoder that checks the data and is capable of correcting a single error.

This scheme has the advantages of a smaller area impact due to the fact that there is no need for triplicating the registers. The main disadvantage of this technique is that it is placed in the datapath and therefore it may increase the critical path delay, thus reducing the maximum frequency at which the circuit may operate. For this work and to show how a pre-designed block could be used to implement this technique a single module, named *Hamming Block*, containing the three blocks (encoder, register, decoder) was developed. This block can store up to 26 data bits with four additional parity bits. It can be easily used by just instantiating it in the RTL code and connecting the signals that need to be stored in a register. The output of this block is comprised of the protected data signals.

The implementation of the Hamming encoding is the following. First, all single-bit signals and buses that are used as register inputs are identified. These signals to be stored are routed to the Hamming block and the signals that are to be "retrieved" are connected to the output of the same Hamming block. The signals are collected in groups of 26 bits, since that is the size if the pre-designed block. The Hamming block is then instantiated for each group of 26 bits. The clock and reset

signals are also connected as to the Hamming Block control signals.

IV. SEU Injection Tool

In order to test and validate the correct implementation and the robustness of the supported techniques, an *SEU injection tool* was developed. This tool was designed to work on top of the SimVision software, and follows the same principles as similar tools developed for a different platform [12], [13]. This tool generates a command script for the simulator that mimics the effect of an SEU occurring in the design. The tool is capable of inserting SEUs in Verilog RTL descriptions and in Verilog gate netlists, this it can be used before and after synthesis. Using the SIT, instead of a specific testbench with explicit SEU bit flip commands, allows the designer to maintain just one testbench, the one used for functional validation. And by using the same testbench for functional verification and to check SEU robustness the designer is sure that the exactly the same stimulus are applied during the tests. And so the detection of an SEU propagating through a design is simplified, since a golden simulation is already available from the functional verification.

The tool developed is to be used in the SimVision simulation/debugging environment. It uses SimVision commands to search for register definitions (in the case of RTL code being simulated) or flip-flop definitions (when doing post synthesis simulation) and creates a list with them all. This list is then used by the tool to generate a sequence of commands in order to mimic an SEU during the simulation. This list can be edited by the designer in order to define which specific registers/nodes should be upset while running the tool. Information about when, in the simulation, the SEU should occur can be also added in the list. The designer is also able to select different modes of SEU injection, by specifying the number of SEUs, the time between SEUs, and if the SEU should be inserted randomly or as defined on the list. After setting all the parameters, the SIT can be executed, and the SEU injection is performed according to the pseudo-code presented in Algorithm 1.

Algorithm 1 Pseudo-code of the SEU injection procedure in SimVision

```
list_reg ← Register list

N ← number of SEU ocurrences

while N > 0 do
    N ← N − 1;
    RUN some defined time;
    Selected_Reg ← random element from list list_reg;
    DEPOSIT Selected_Reg ← !value.Selected_Reg;
end while
RUN until the end
```

The tool collects information about the registers that are to be upset, and also the time when the event should occur from the parameter files, and then runs the simulation until the time for the first SEU. At that moment, it deposits the complement of the current value in the register that is to be upset. The same procedure is repeated until the total number of SEUs has been reached or the simulation ends.

The tool creates a log file with information about when SEU injection has occurred, in which register and to which value the register was upset. A TCL command file is also generated, which contains the commands used to simulate the SEU injection. This command file allows the designer to run the same stimulus later, applying the same SEU bit flips. This is useful to debug any SEU robustness issue that was found in the first run and to confirm later that it has indeed been solved.

V. Implementation and Evaluation

The techniques previously presented were applied to several Verilog modules, namely two filters and a command decoder. The first filter,*Filter 1*, is a Finite Impulse Response filter with an input of 9 bits and a output of 18 bits width[14]. It has a latency of five clock cycles due to the internal stages. The filter has also a reset signal that places the output and all the internal registers at zero. The second filter, *Filter 2*, is based on the one presented in[15], which is a tail cancellation filter used as a component in a front-end electronic system in the ALICE detector at CERN. This filter receives an 18-bits input and produces an 18-bit output. The filter has three stages where multipliers and adders are used to compute the output. The command decoder, is used to interpret variable-length commands that are received in serialized form through a 1-bit input port. The decoded command is then used to control the behaviour of 50 output signals. This block is also responsible for reading some internal registers of a system; for that it has additional 22 input signals and three 16-bit parallel bus inputs. A reset input signal is also present.

The development of these circuits and theirs testbenches is outside the scope of this paper. For each circuit, both the description and the testbench are provided by the respective design teams. In fact the goal of this workflow is to work with already developed circuits. The modules were simulated and validated before the implementation of the protective techniques was added. The circuits were then tested with the SIT in order to verify the susceptibility of the circuit to SEU bit-flips. This procedure may be used to identify which parts of a design should be better protected, or not protected at all.

In these cases the same protective technique was applied to the entire Verilog module and all the modules where protected independently with the three protective techniques presented earlier. The reasons for this choice are: (i) these modules belong to a low level of the system's module hierarchy, and (ii) this choice enables a better extraction of meaningful comparative values for each technique. The triplication and the Hamming code implementation were done by running the scripts that introduce the protective techniques in each circuit. Next the circuits were simulated again using the same testbenches as before, in order to verify that the functionality was preserved. After verifying the functionality, one more simulation done with the SIT was executed to verify the robustness of the circuit against SEU bit-flips.

Table I presents the results for all of the implementations under evaluation after RTL synthesis and mapping to a 130 nm CMOS library. The table presents the power consumption estimates for each implementation of the circuit. The number of logic gates used is presented as well as the leakage, dynamic and total power consumption. Comparative percentage from the

TABLE I. POWER USAGE BY TECHNIQUE AND RELATIVE PERCENTAGE

Module name	Logic gates	Leakage power (µW)	Dynamic power (µW)	Total power (µW)
Filter 1 Original	1928	308	1079	1387
Filter 1 Hamming	5531 (187 %)	703 (128 %)	4142 (284 %)	4845 (249 %)
Filter 1 TMR1	7528 (290 %)	1038 (237 %)	4095 (280 %)	5134 (270 %)
Filter 1 TMR2	6870 (256 %)	1005 (226 %)	3471 (222 %)	4476 (223 %)
CD Original	648	108	310	419
CD Hamming	1684 (160 %)	245 (127 %)	991 (220 %)	1236 (195 %)
CD TMR1	2524 (290 %)	343 (218 %)	789 (155 %)	1133 (170 %)
CD TMR2	2528 (290 %)	334 (209 %)	754 (143 %)	1088 (160 %)
Filter 2 Original	1489	178	1217	1395
Filter 2 Hamming	2592 (74 %)	304 (70 %)	1970 (62 %)	2275 (63 %)
Filter 2 TMR1	5105 (243 %)	571 (221 %)	4063 (233 %)	4635 (232 %)
Filter 2 TMR2	4977 (234 %)	555 (212 %)	3395 (179 %)	3950 (183 %)

original implementation is also added for a better understanding of the overhead introduced by each technique.

From the table is possible to conclude that the Hamming approach requires a smaller number of logic gates (minimum increase of 74 % and a maximum of 187 %) when compared with the triplication (minimum 234 % to maximum 290 %). On the other hand the dynamic power of the Hamming technique increases more than in the case of triplication. 284 % (Hamming) against 250 % (triplication) of increase in dynamic power for benchmark *Filter 1* and 220 % (Hamming) against 155 % (triplication) of increase in dynamic power for benchmark *CD*. This increase in dynamic power has to do with the fact that the combinational part of the encoder and decoder of the Hamming technique add a significant number of logic gates to the design, which are active in every clock cycle (in order to compute the parity bits). The comparative analysis between the technique TMR 1 (simple triplication) and TMR 2 (chain triplication), shows that chain triplication has a smaller impact in the number of logic gates used (15 % less), as well as smaller power consumption (35 % less), when compared with the simple triplication approach. However not all the data follow the same trend and for instance the data from the Hamming implementation for benchmark *Filter 2* has actually a lower increase in both number of logic gates and as well in the dynamic power (74 % for logic gates, and only 62 % for dynamic power overhead) when compared with the same technique on a different benchmark. This is an evidence that a one-solution approach to protect a complex system may not be very efficient in terms of are and power consumption.

Table II shows the distribution of the leakage power consumption and relative area usage by the type of gates, either sequential or combinational gates. In this table it is possible to observe the changes in the amount of gates in each domain. For instance, in the case of Hamming encoding, the balance between sequential and combinational gates is altered considerably from almost 50 % to 76 % of area occupied with combinational logic. While for the triplication the difference between the original and the triplicated version did not change as much, it changed from as little as 52 % of area occupied by combinational gates to only to a maximum of 62 % for the benchmark *Filter 1*. *Filter 2* benchmark has almost no change on the balance between combinational and sequential when comparing the original and the triplication. Still for the same table is possible to observe that an increase in the relative area occupation of the combinational part for a given circuit has also an increase the the leakage power consumption. However the percentage for the relative area occupation may not be precisely the same as the one for leakage power due to the combinational part. For example, the benchmark *Filter 1* for the TMR2 implementation has 61 % for relative area occupation but the impact of the combinatorial logic in the overall leakage power consumption is only of 36 %.

VI. CONCLUSION AND FUTURE WORK

This paper presented a workflow for improving the SEU robustness of digital circuits described at the register transfer level. A description of each procedure was discussed and the objectives of each were explained. The advantages and disadvantages of each SEU protection technique supported by the tools were also presented and discussed. Finally, the implementation of those techniques was applied to several digital modules, and the usability of the flow was evaluated.

The evaluation shows that the implementation of SEU robustness can be made in an early stage of the design flow (before logic synthesis), and it can be applied directly to a digital module that was not designed taking into account the SEU robustness. However the flow can also be used for designs presented as Verilog gate netlist. The time that it takes to go through the described work flow is reduced when compared to the approach where the SEU protection is implemented during the design of the module itself (at the same time as the main functionality). The *SEU injection tool* (SIT) together with the reuse of the functional testbench proved to be a good combination that allows the designer to check which parts of the circuit are critically affected by SEU bit flips. By reusing the testbench that validates the functional aspect of the design, the designer is also reassured that the functionality of the final circuit is intact.

Future work will expand the workflow in order to support other protective techniques against SEUs. For example, one addition would be to support the automated recoding of the state variables of a state machine. A preliminary analysis has shown that encoding the state variables in order to reduced the SEU sensitivity and to lower the power consumption may be a useful

TABLE II. AREA AND POWER DISTRIBUTION AMONGST TECHNIQUES

		Sequential Gates			Combinational Gates		
Circuit	Technique	Number of instances	Relative area occupation %	Leakage power %	Number of instances	Relative area occupation %	Leakage power %
Filter 1	Original	496	40.3	64	1010	52.6	31.1
CD	Original	165	46	70.8	419	49.8	26.3
Filter 2	Original	162	16.7	42.8	1040	77.4	52.1
Filter 1	Hamming	644	19.6	36.4	4085	75.4	59.7
CD	Hamming	205	20.7	33.2	1330	76	64.4
Filter 2	Hamming	207	12.8	27.1	2030	82.5	69
Filter 1	TMR1	1488	31.1	56.9	4649	62.7	38
CD	TMR1	495	34	57.2	1604	58.1	32.9
Filter 2	TMR1	486	14.5	33.8	3666	79.5	59.9
Filter 1	TMR2	1488	32.7	58.8	3984	60.9	36.4
CD	TMR2	495	33.9	59	1613	58.3	31.4
Filter 2	TMR2	486	14.6	35.2	3565	79.4	58.4

technique to support. Another planned improvement is the determination of area and power consumption information from post-layout data. A more precise and accurate data regarding the overhead caused by adding protection against SEUs might prove very usefull for increase the efficiency of the techniques used in a project. The efficiency is gained in the sense that the designer with proper and acurate data can choose the best protection technique to its needs. As for now the extraction of the data regarding the sensitivity of a design to SEU, data produced with the SIT, needs to be done manually by inspecting the testbench result or by using the *SimVision compare* tool. Another planned improvement is to produce a summary containing the most sensitive nodes, the ones that if hit by an SEU will propagate the error through the design.

ACKNOWLEDGMENT

This work is supported by CERN and the ATLAS Experiment doctoral student program, and by the ERDF – European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundao para a Ciłncia e a Tecnologia (Portuguese Foundation for Science and Technology) within project ”FCOMP - 01-0124-FEDER-022701”.

REFERENCES

[1] F. Faccio, “Design hardening methodologies for ASICs,” in *Radiation Effects on Embedded Systems*, R. Velazco, P. Fouillat, and R. Reis, Eds. Springer Netherlands, Jan. 2007, pp. 143 – 160.

[2] R. Naseer and J. Draper, “DF-DICE: a scalable solution for soft error tolerant circuit design,” in *Proc. 2006 IEEE Intl. Symposium on Circuits and Systems*, 2006, p. 4 pp.

[3] S. Jahinuzzaman and R. Islam, “TSPC-DICE: a single phase clock high performance SEU hardened flip-flop,” in *53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2010*, Aug. 2010, pp. 73 – 76.

[4] M. Fazeli, A. Patooghy, S. Miremadi, and A. Ejlali, “Feedback redundancy: A power efficient SEU-Tolerant latch design for deep sub-micron technologies,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007.*, Jun. 2007, pp. 276 – 285.

[5] J. Rockett, L.R., “An SEU-hardened CMOS data latch design,” *IEEE Transactions on Nuclear Science*, vol. 35, no. 6, pp. 1682 – 1687, 1988.

[6] R. Oliveira, A. Jagirdar, and T. J. Chakraborty, “A TMR scheme for SEU mitigation in scan flip-flops,” in *8th International Symposium on Quality Electronic Design, 2007. ISQED ’07*, Mar. 2007, pp. 905 –910.

[7] P. K. 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.

[8] W. Chen, R. Gong, K. Dai, F. Liu, and Z. Wang, “Two new space-time triple modular redundancy techniques for improving fault tolerance of computer systems,” *In IEEE International Conference on Computer and Information Technology*, p. 175, 2006.

[9] S.-F. Liu, P. Reviriego, and J. Maestro, “Fault tolerant FIR filters using hamming codes,” in *2009 European Conference on Radiation and Its Effects on Components and Systems (RADECS),*, Sep. 2009, pp. 493 –496.

[10] J. A. M. O. Ruano, “A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs,” *IEEE Transactions on Nuclear Science*, no. 4, pp. 2091 – 2102, 2009.

[11] J. K. Ousterhout, “Tcl: An embeddable command language,” 1990, pp. 133 – 146.

[12] Electronic Design and Space Technology group at Universidad Antonio de Nebrija. (2005) SEUs simulation tool. [Online]. Available: http://www.nebrija.es/~jmaestro/esa/sst.htm

[13] A. Benso, S. Martinetto, P. Prinetto, and R. Mariani, “A SEU injection tool to evaluate DSP-based architectures for space applications,” in *Proceedings 2000 International Conference on Computer Design,*, 2000, pp. 537–538.

[14] F. Kastensmidt, L. Sterpone, L. Carro, and M. Reorda, “On the optimal design of triple modular redundancy logic for SRAM-based FPGAs,” in *Proceedings on Design, Automation and Test in Europe, 2005.*, 2005, pp. 1290 – 1295 Vol. 2.

[15] B. Mota and D. Mlynek, “Time-domain signal processing algorithms and their implementation in the ALTRO chip for the ALICE TPC,” Ph.D. dissertation, Ecole Polytechnique, Lausanne, Geneva, 2003, presented on 26 May 2003.