Analyzing Area and Performance Penalty of Protecting Different Digital Modules with Hamming Code and Triple Modular Redundancy

R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, R. Reis

Universidade Federal do Rio Grande do Sul
PPGC - Instituto de Informática - DELET
Caixa Postal 15064. CEP 91501-970 - Porto Alegre - RS - Brasil
e-mail: <renato, felipem, fglima, carro, susin, reis>@inf.ufrgs.br

Abstract

This work compares two fault tolerance techniques, Hamming code and Triple Modular Redundancy (TMR), that are largely used to mitigate Single Event Upsets in integrated circuits, in terms of area and performance penalty. Both techniques were implemented in VHDL and tested in two target applications: arithmetic circuits with pipeline and registers files. Area overhead results show that TMR is more appropriated for modules using single registers like in pipelines, control and datapath circuits, while Hamming code is a better trade-off for groups of registers, such as register files, caches and embedded memories.

1. Introduction

Integrated Circuits operating under radiation may be affected by undesirable effects caused by charged particles located in the space environment [1]. Radiation effects can be classified in Total Ionizing Dose (T.I.D.) and Single Event Effects (S.E.E.). T.I.D. defines a long-term degradation of electronics due to the cumulated charge deposited in the silicon. S.E.E. is a transient effect induced by the trespassing of a single charged particle through the silicon. In addition, due to the constant shrink in the transistor dimensions, particles that once were considered negligible now are significant to cause upsets. [2]

When a single charged particle strikes the silicon, it loses its energy via the production of electron hole pairs resulting in a dense ionized track in the local region. This event can provoke a current pulse, which can perturb the integrated circuit operation. This work targets the Single Event Upset (SEU) phenomenon that is defined by a random bit inversion of a storage cell. Targets are registers, memories and all latches and flip-flops in general.

SEU protection techniques must consider all storage cells in order to achieve full reliability in the system. Examples of SEU mitigation techniques are Triple Module Redundancy (TMR) [3], Error Detection and Correction Code (EDAC), like Hamming Code [3], and hardened memory cells [4]. Each one of them has some advantages and drawbacks, and it can be more suitable for some specific architecture. The objective of this work is to find the most cost efficient approach analyzing area overhead, performance penalty and SEU tolerance.

This work aims to investigate the advantages and drawbacks of using the TMR and Hamming code techniques in two target applications: FIR filters and register files. These applications were chosen because of their large usage in present days systems. In the case of the filter, pipelines are required to increase system throughput. Pipelines are composed of registers. For each pipeline stage one needs to include one coder and decoder (for Hamming protection), or one voter (for TMR protection). Modules composed of register banks and embedded memories were also investigated under TMR and Hamming code protection. Results were analyzed in both structures in terms of area and delay overhead.

The paper is organized as follows. Section 2 presents some brief comments about related works. Section 3 introduces the SEU mitigation analysis. In this section the Hamming code and TMR are discussed in terms of implementation, area overhead and performance penalty. Section 4 shows two studied cases: the pipelined FIR filter and the register files. Main conclusions are presented in section 5.

2. Related Work

Many commercial microprocessors from Intel, IBM, Motorola and Sun are available in the market in a radiation tolerant version. These microprocessors are designed and protected by space project companies and research laboratories. Each product offers different levels of radiation immunity for distinct space and military applications. The techniques used to protect the microprocessors are usually based on the process technology or package shielding, TMR, SEU hardened memory cells, EDAC (Hamming code) or a combination of them.

It is common to find microprocessor with the memory protected by Hamming code. One example is the Atmel SPARC microprocessor that is protected by EDAC [5]. Another example is the SEU tolerant

Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02)
0-7695-1807-9/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

microprocessor PowerPC from Motorola designed by Maxwell [6] where the memory is also protected by EDAC. In this last case, the CPU is also protected using the TMR technique. The TMR was implemented in the full CPU block, instead of only in the registers and storage cells in general. The voter compares the output of each of the CPUs on a bit-by-bit basis. Another example is the SEU tolerant 8051 micro-controller presented in [7, 8] where all registers and the internal memory were protected by hamming code.

Programmable logic companies also uses SEU mitigation techniques in order to provide radiation tolerant versions of their devices. One example is Xilinx that uses Triple Modular Redundancy (TMR) with voting technique combining with bitstream scrubbing to protect the Virtex FPGAs [9].

The challenge is to find the most appropriate SEU mitigation technique for each structure in the circuit concerning reliability, area and performance.

3. SEU Mitigation Techniques Analysis

The first SEU mitigation solution that has been used for many years in spacecraft is shielding. It reduces the particle flux but it does not completely eliminate it. Consequently, extra techniques must be applied to avoid SEU. The SEU mitigation techniques used nowadays in the logic level are basically based on Triple Module Redundancy (TMR), Hamming code and hardened memory cells.

In the first technique, all memory cells are triplicated and a voter is added to the memory cell outputs. It is possible to restore one error at each bit. In best case, n errors in a n bits word can be tolerated. The second technique is a parity code, where one has a larger word (approximately $n + \log_2(n)$ bit words for n bits of information) and can restore one bit error. There are codes capable of restoring two or more bit errors, but they are not discussed in this work. The third technique is based on replacing all storage cells by a hardened cell composed of extra transistors able to avoid the occurrence of a bit flip (upset).

Any chosen protection technique will mean an area and performance penalties. TMR triplicates all the storage cells area, and adds voters in the outputs. Hamming codes increase the number of storage cells because of the parity bits (approximately by $\log_2 n$) and coder and decoder logic blocks are needed. Both of them add combinatory logic to the critical path, so the circuit frequency may be decreased. Hardened storage cells increase the area of each cell and it can increase the read and write time of the storage cell.

3.1. Hamming Code Implementation Analysis

Hamming code is an error-detecting and error-correcting binary code that satisfies the equation, $d+p+1 \leq 2^p$, where d is the number of data bits and p is the number of parity bits. Following this equation the Hamming code can correct all single-bit errors on *d*-bit words and detect double-bit errors when an overall parity check bit is used (SEC-DED) [3]. The hamming code implementation is composed of a combinational block responsible to code the data (encode block), extra bits in the word that indicate the parity (extra latches or flip-flops) and another combinational block responsible to decode the data (decode block).

The coder block calculates the parity bits. The steps to create the code word are described as follow. First mark all bit positions that are powers of two as parity bits (1,2,4,8,16, etc). Each redundant bit P is the parity of a set of bits in the word. All other bit positions are for the data to be encoded (3,5,6,7,9, etc). Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips. Position 1 checks 1 bit and skips 1 bit (1,3,5,7,9,11, etc). Position 2 checks 2 bits and skips 2 bits (2,3,6,7,10,11, etc). Position 4 checks 4 bits and skips 4 (4,5,6,7,12,13, etc). Position 8 checks 8 bits and skips 8 bits (8-15,24-31, etc).

The coder block can be implemented by a set of 2-input XOR gates. For an 8-bit data, 14 2-input XOR gates are necessary in order to generate the 4 parity bits, while for a 4-bit data, only 3 2-input XOR gates are necessary to generate the 3 parity bits, as illustrated in figure 1.

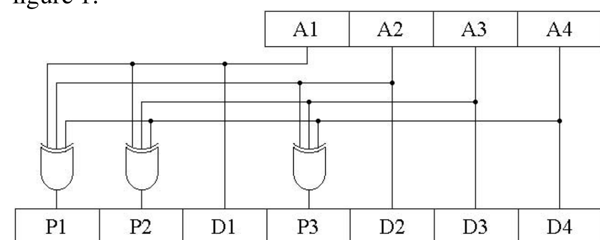


Figure 1 – Hamming coder architecture

The decoder block is more complex than the coder block, because it needs not only to detect the fault, but it must be able to correct it. It is basically composed of the same logic used to compose the parity bits plus a decoder that will indicate the bit address that contains the upset. The decoder block can also be composed of a set of 2-input XOR gates and some AND and inverter gate.

If all parity bits are 0 the word is correct. If at least one of the parities is 1, there is a bit inversion. The inverted bit position is calculated by concatenation from P3P2P1 and reading it as a unique binary number. For example, considering a 4 bit data, we want to code number 6 (0110). Its coding is 1100110. Introducing a bit flip we have: 1100010. The following parity bits are calculated P1 = 1, P2 = 0, P3 = 1. Reading P3P2P1 = 101 we realize that the error is in position 101, or 5 in decimal. Figure 2 shows the VHDL code of a 4-bit hamming decoder.

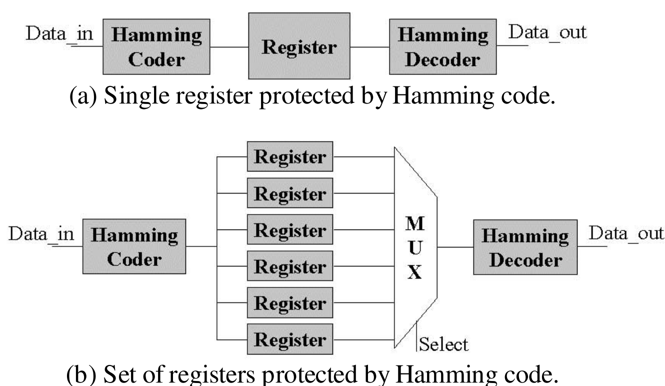
IEEE COMPUTER SOCIETY

```
-- First Step: Parity bits generation
par(2) <= data_in(0) xor data_in(2) xor
          data_in(4) xor data_in(6);
par(1) <= data_in(1) xor data_in(2) xor
          data_in(5) xor data_in(6);
par(0) <= data_in(3) xor data_in(4) xor
          data_in(5) xor data_in(6);
-- Second Step: Error Verification and Correction
mask <=
    "1000000" when par = "111" else
    "0100000" when par = "110" else
    "0010000" when par = "101" else
    "0001000" when par = "100" else
    "0000100" when par = "011" else
    "0000010" when par = "010" else
    "0000001" when par = "001" else
    "0000000";
coded <= data_in xor mask;
decoded_out (0) <= coded(2);
decoded_out (1) <= coded(4);
decoded_out (2) <= coded(5);
decoded_out (3) <= coded(6);
```

Figure 2 – Hamming decoder description

The applied coding method is recommended for systems with low probabilities of multiple errors in the same coded word. Hamming code can also correct double bit errors (SEC-DEC), however the algorithm is much more complex and it is not in the scope of this work.

Structures such as registers, registers file and memories can be protected by hamming code. Each protected register must have its input connected to a coder block and its output connected to a decoder block, as shown in figure 3a. The register file example is given in figure 3b. Note that only one register may be used at a clock cycle. The main advantage of the set of registers structure is that only one coder and decoder logic are multiplexed for a set of registers.



(a) Single register protected by Hamming code.

(b) Set of registers protected by Hamming code.

Figure 3 – Hamming code implementations

Hamming code increases area by requiring additional storage cells, plus the coder and the decoder blocks. For an *n* bit word, there are approximately $\log_2 n$ more storage cells. However, coders and decoders may add a more significant area increase. Table I shows area values for coders and decoders. The blocks were described in VHDL and synthesized in two different FPGAs: Flex10K20 by Altera [10] and XCV300 by Xilinx [11]. For an 8-bit word, 4 parity bits are needed for the coded word, while for a 16-bit word, 5 parity bits are needed.

The delay of the coder and decoder block is added in the critical path as presented in figure 4. Table I show the delay of the coders and the decoders synthesized to the respective FPGAs. Notice that the delay gets more critical when the number of bits in coded word increases. The number of XOR gates in serial is directly proportional to the number of bits in the coded word.

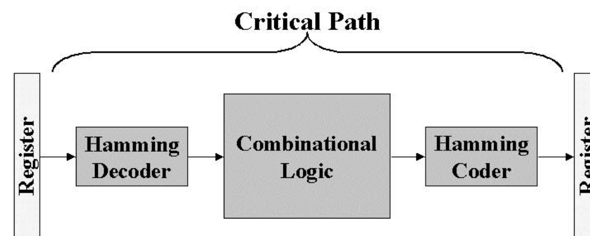


Figure 4 –Hamming code critical path

Table I - Coders and decoders area results

Design	Flex10K20		XCV300	
	#LCs	Delay (ns)	#CLBs	Delay(ns)
8-bit Coder	14	13.9	2	12.18
8-bit Decoder	24	26.6	8	16.46
16-bit Coder	29	16.7	10	15.54
16-bit Decoder	51	40.7	19	21.52

3.2 TMR Implementation Analysis

Triple Module Redundancy (TMR) [3] is a very common fault tolerance technique. This technique can be used to protect circuits against radiation effects. The principle is: triplicate the hardware and add a voter in the outputs. In this paper we are only using TMR in memory elements. All memorization elements are tripled and its respective outputs are connected to a voter. The voter will select the output of the majority of the components. So, if one component fails, the error will not be reflected in the voter output. The voter is implemented by few logic gates, for each bit, as it can be seen in figure 5.

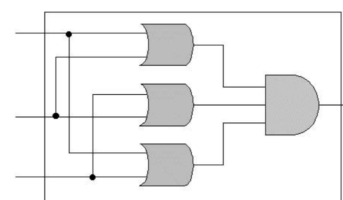


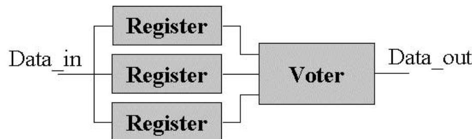
Figure 5 – Voter architecture.

To protect circuits against radiation effects, all registers must be tripled. The voter must be at their output. Figure 6 shows TMR protecting a single register (figure 6a) and a set of registers (figure 6b).

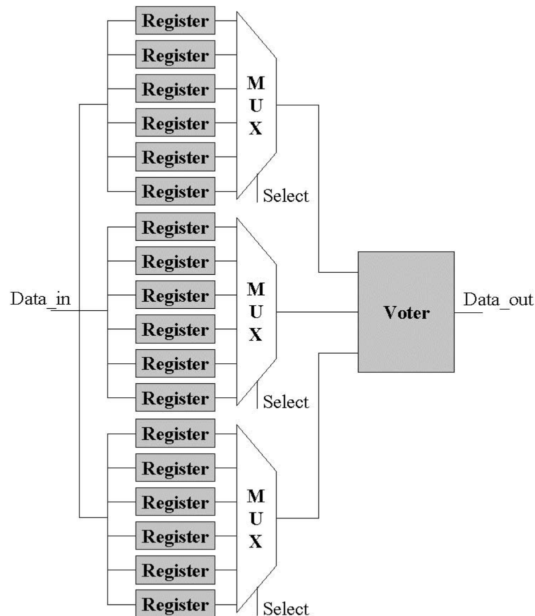
The area increase of TMR is the addition of two registers of the same size, i.e. each memory cell is

0-7695-1807-9/02 $17.00 © 2002 **IEEE**
IEEE COMPUTER SOCIETY

replaced by a set of three memory cells. Besides, there are n voters for each n bit register. Table II shows the voter area addition, synthesized in two different FPGAs: the Flex10K20 by Altera [10] and XCV300 by Xilinx [11].



(a) Single register protected by TMR.



(b) Set of registers protected by TMR.

Figure 6 –TMR implementations

In the circuit critical path there will be a voter, as it can be seen in figure 7. This shows that the clock period will be added by the voter delay. Table II shows delay values for two voter implementations.

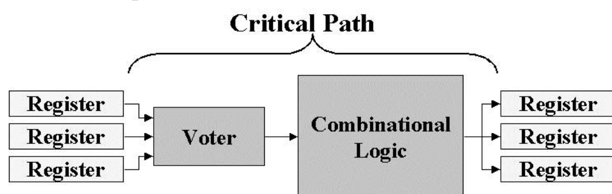


Figure 7 – TMR critical path

Table II – TMR voters area and delay results

Design	Flex10K20		XCV300	
	#LCs	Delay (ns)	#CLBs	Delay (ns)
8-bit Voter	8	13.8	8	13.53
16-bit Voter	16	13.8	16	15.60

3.3 Comparison between TMR and Hamming Code Techniques

The storage elements are distributed in a circuit as registers of different sizes or as bank of registers such as caches, register files and embedded memories. In order to compare the efficiency of TMR and Hamming code in terms of area overhead, two cases are considered for implementation, single registers of different sizes and register files of different sizes.

In the first case, each register is protected and have its own coder/decoder or voter. Figure 8 shows graphically the comparison between Hamming and TMR protected register with sizes ranging from 1 to 32 bits. Note that TMR have less area. The overhead of the Hamming code over the TMR increases proportionally to the size of the protected word. This result is supported by the fact that the coder and decoder logic strongly increase with the size of the word to be protected. In addition, the Hamming code delay overhead is also bigger because of the large number of XOR gates in serial located in the code and decode logic.

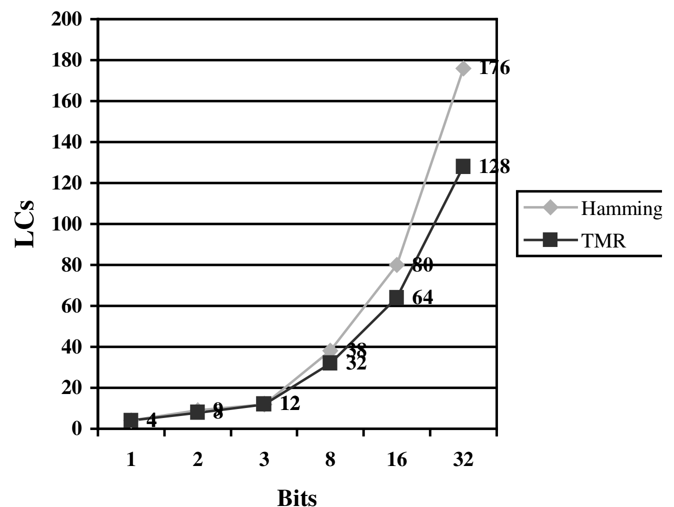


Figure 8 –Single protected registers

In the second case, groups of registers are protected sharing the same coder/decoder or voter. In this case, the area increase of memory cells is more significant then coder/decoder area because one coder/decoder is shared to many registers. Figure 9 shows the Hamming Code superiority for that. The Hamming code in this case is more efficient in terms of area than the TMR. Both table II and Figure 8 show that TMR has linear progression, while Hamming do not. In figure 8, TMR progression is 4 LCs for each bit (three Flip-Flops and a Voter). In figure 9, TMR progression is 23 (15 FFs, 1 voter and the rest for Multiplexors).

IEEE COMPUTER SOCIETY

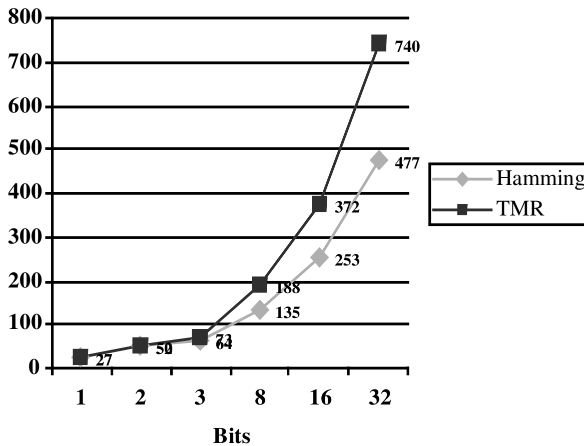


Figure 9 –Multiple protected registers

4. Case Studies: FIR Filter and Register File

This work employs two example applications: a FIR filter and register file to evaluate area and delay overhead of using Hamming code and TMR. The two techniques were implemented in VHDL in each application.

4.1 FIR Filter

The FIR filter architecture is shown in figure 10. It is a six tabs filter. In this approach we are using 8-bit data registers (R1, R2, R3, R4, R5 and R6) for each tap of the filter. The filter coefficients were calculated from MatLab [12]. All coefficients were rounded for an 8-bit representation. The multipliers are optimized to multiply constant values. They were automatically generated by the Lemon Dragon Multiplier Generator [13].

Table III shows synthesis results for pipelined filter in the Flex10K20 from Altera [10] and the XCV300 from Xilinx [11]. The Hamming protected filter is larger and slower than the TMR one in both technologies. Note that each register (including pipeline registers) has one coder and one decoder as the first case presented in session 4. This reflects the results shown on table III.

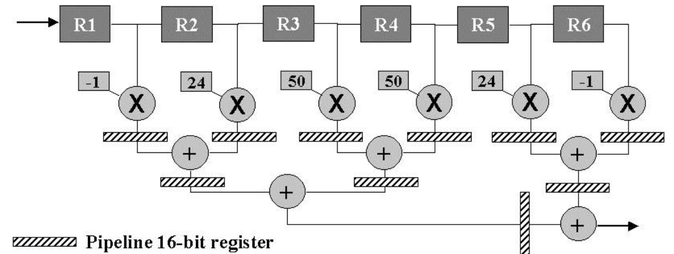


Figure 10 – FIR filter topology

4.2 Register File

The Register file is a VHDL module with an address, a data and a write signal as input. The output is the data of the register pointed by the address. Table IV shows synthesis results for a five registers bank. Results show that the use of only one coder and one decoder block for the whole group of registers has improved the results of the Hamming code compared to the TMR in the case of registers file.

4.3 Final remarks

Table V shows an evaluation summary of Hamming code and TMR. For small bit words, Hamming code and TMR are equivalent in terms of area overhead and performance penalty.

Notice that both solutions are not able to correct the upset in the storage cell, only to propagate the correct value if an upset occurs. The extra logic used for correction can be composed of a return path with multiplexor, for example. Concerning not only single upsets but multiple upsets in the protected word, TMR can vote the correct value when any number of upsets occur in distinct bits. On the other hand, Hamming code can only correct single bit upsets. In order to correct more than one single upset, more complex error correction code algorithms must be used.

Table III – Implementation Results of FIR Filters

Design	# ffs	Flex10K20		XCV300	
		#LCs	Delay (ns)	#CLBs	Delay(ns)
Pipelined FIR filter	176	385	52.3	217	35.48
Protected by Hamming	240	1120	94.1	482	41.73
Protected by TMR	528	894	61.0	477	40.77

Table IV - Coders and decoders Area Results

Design	# ffs	Flex10K20		XCV300	
		#LCs	Delay (ns)	#CLBs	Delay(ns)
8 bit Register File without protection	40	79	25.3	37	16.36
8 bit Register File with Hamming	60	142	36.6	63	25.11
8 bit Register File with TMR	120	191	26.8	105	21.64
16 bit Register File without protection	80	151	28.4	69	21.98
16 bit Register File with Hamming	105	253	63.6	115	35.51
16 bit Register File with TMR	240	372	23.8	205	27.60

Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02)
0-7695-1807-9/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

Table V – Hamming Code and TMR Comparison Summary

	Hamming Code (SEC-DED)	TMR
Area	It depends on the number of bits to be protected. It has a small overhead of storage cells, and additional combinational logic to implement the coder and the decoder logic blocks.	It needs 3 times more storage cells and small extra logic for the voters. The number of voters is proportional to the number of storage cells.
Performance	The performance can be affected by the coder and decoder blocks that are located in the critical path. The delay increases proportionally to the number of bits to be coded because of the number of XOR gates in serial in the code/decode blocks.	The performance is not strongly affected because the only source of delay is the voter that is basically constant with the number of bits to be protected.
Error correction	It corrects one single upset per word. But it does not correct the upset in the stored word. Upsets will accumulate if there is no extra logic to correct them.	It corrects up to n upsets per n-bit word if each upset is located in a distinct bit. It votes the correct value but it does not correct it. Upsets will accumulate if there is no extra logic to correct them.

5. Conclusions

This paper has analyzed the area and performance impact of protecting a single register or a group of them with Hamming code and TMR. Results from the two studied cases show the advantages and drawbacks of each method. TMR increases significantly the area of memory cells, while the voter is implemented with a small number of logic gates. The number of voters increases linearly with the number of bits in the protected word. On the other hand, Hamming code produces a small increase in the number of storage cells, but it needs large coders and decoders logic blocks, which considerably increase the area and the delay of the critical path. Results showed that Hamming is a better technique in terms of area to be applied in memories and register files for both FPGA implementations (Flex10K20 and XCV300), while the TMR technique is more advantageous in terms of area to be applied in registers and single or small group of storage cells.

The delay of the Hamming code is always worse than the TMR delay, because voters are implemented with 1 LC or and AND/OR gate structure, while Hamming coders and decoders, both located in the critical path, have multi-level XOR gates. Even when Hamming code is the better choice in terms of area to be used in memories or register files, it can be inappropriate in some cases, when the large number of bits will provoke a long path of serial XOR gates in the coder and in the decoder block. In this case, it is necessary to partition the circuit as presented in [15] in order to code and decode small bit words. Note that for words up to 16 bits, the difference in area and delay between hamming code and TMR is almost insignificant.

The main conclusions of this work is that TMR is an appropriate technique for blocks composed of pipelines and individual registers in general, while Hamming is suitable to protect register files and embedded memories, as long as the words are not too large and the extra delay is acceptable.

6. References

[1] BARTH, J., “Radiation Environment,” IEEE Nuclear and Space Radiation Effects Conference Short Course, Jul. 1997.

[2] JOHNSTON, A. “Scaling and Technology Issues for Soft Error Rates”, 4th Annual Research Conference on Reliability, Stanford University, Oct. 2000.

[3] PETERSON, W., “Error-correcting codes,” 2nd ed., Cambridge : The MIT Press, 1980. 560p.

[5] VELAZCO, R.; BESSOT, D.; DUZELLIR, S.; ECOFFET, R.; KOGA, R. Two Memory Cells Suitable for the Design of SEU-Tolerant VLSI Circuits. In: IEEE Transactions on Nuclear Science. VOL. 41, NO. 6, December 1994.

[6] ATMEL, “Rad-Hard Embedded Processor 32-bit SPARC,” TSC695F, Data Sheet, www.atmel.com, March 2001.

[7] MAXWELL TECHNOLOGIES. Datasheet. Availble for download at http://www.spaceelectronics.com/documentation/datasheets.html (Nov. 2001)

[8] LIMA, F., REZGUI, S., COTA, E., CARRO, L., LUBASZEWSKI, M., VELAZCO, R., REIS, Ricardo, “Designing and Testing a Radiation Hardened 8051-like Micro-controller”, MAPLD Conference, Sept. 2000.

[9] COTA, E.; LIMA, F.; REZGUI, S.; CARRO, L.; VELAZCO, R.; LUBASZEWSKI, M.; REIS, R. “Synthesis of an 8051-like Micro-Controller Tolerant to Transient Faults”, JETTA, 2001.

[10] CARMICHAEL C. Triple Module Redundancy Design Techniques for the Virtex TM Series. Xilinx Application Note xapp197, 2001.

[11] ALTERA. FLEX 10K Embedded Programmable Logic Family Data Sheet. V. 4.1, March 2001.

[12] XILINX, INC. Virtex™ 2.5 V Field Programmable Gate Arrays, Xilinx Datasheet DS003, v2.4, Oct. 2000.

[13] MathWorks. MATLAB: the Language of Technical Computing. MathWorks, Natick, MA, 1997.

[14] HENTSCHKE Renato, CARRO Luigi, REIS Ricardo. Lemon Dragon Multiplier Generator. Available For Download at http://www.inf.ufrgs.br/~renato/download

[15] HOLLANDER, H., et al., Synthesis of SEU – Tolerant ASIC Using Current Error Correction, Proceedings of the fifth Great Lakes Symposium on VLSI, pp. 90-93, March 1995.

0-7695-1807-9/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY