Revista Brasileira de Computação Aplicada, July, 2020

DOI: 10.5335/rbca.v12i2.10120
Vol. 12, Nº 2, pp. 93–102
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ORIGINAL PAPER

Evaluation of a Network-on-Chip designed to deal with multiple processors in a nanosatellite

Liz Cristine Moreira Coutinho[1] and Marcelo Daniel Berejuck [2]

[1,2]Federal University of Santa Catarina - UFSC

[*]lizcmoreira@hotmail.com; marcelo.berejuck@ufsc.br

Received: 2019-10-23. **Revised:** 2020-06-05. **Accepted:** 2020-06-30.

Abstract

Nanosatellites are part of a category of artificial satellites with two essential characteristics: reduced size and low cost of raw material for their construction. They usually have between four and five boards with embedded electronics, which control all their functions. As an alternative to further minimize its costs is the development of a System-on-Chip or SoC, which may use a commercially available programmable logic device (or COTS – Commercial-Off-The-Shelf component), as an FPGA. To use this type of device is necessary to add mechanisms that can avoid problems arising from cosmic radiation in its electronic components, characteristic in the nanosatellites' space environment. This work presents the proposal of a Network-on-Chip (called NoC) that connects up to four soft-core processors, forming an SoC. The Network-on-Chip, supported by the Hamming codes technique, provides mechanisms that allow it to re-establish a temporary failure of the types Single Event Upset and Single Event Transient. This protection was confirmed by simulations carried out with a software that allows the injection of faults, named ModelSim. Results of lower silicon consumption, concomitant to economies in design and reduction of sensitivity to cosmic ionization, can be observed after the elaborated experiments.

Keywords: Network-on-Chip; Nanosatellite; Hamming Code; Single Event Effect; Single Event Upset; Single Event Transient.

Resumo

Nanossatélites fazem parte de uma categoria de satélites artificiais com duas características importantes: tamanho reduzido e baixo custo de matéria prima para sua construção. Eles, normalmente, possuem entre quatro e cinco placas com eletrônica embarcada, as quais controlam todas as suas funções. Uma alternativa para minimizar ainda mais seus custos é o desenvolvimento de um sistema *intra-chip* (também conhecido como *System-on-Chip* ou SoC), que pode utilizar um dispositivo lógico programável, tipo FPGA, de uso comercial (ou COTS, acrônimo do inglês *Commercial Off-The-Shelf component*). Para utilizar este tipo de dispositivo, é necessário acrescentar mecanismos que possam evitar problemas decorrentes da radiação cósmica que atinge os componentes eletrônicos, o que é comum no ambiente espacial por onde circulam os nanossatélites. Este trabalho apresenta a proposta de uma rede *intra-chip* que conecta até quatro processadores do tipo *soft-core*, formando um SoC. A rede, amparada pela técnica dos códigos de Hamming, oferece mecanismos que permitem a ela seu restabelecimento de uma falha temporária dos tipos *Single Event Upset* e *Single Event Transient*. Esta proteção foi confirmada com simulações realizadas com um *software* que possibilita a injeção de falhas, denominado ModelSim. Resultados de menor consumo de silício, concomitante a economias no projeto e redução da sensibilidade à ionização cósmica, podem ser observados após os experimentos elaborados.

Palavras-Chave: Rede *intra-chip*; Nanossatélite; Código de Hamming; Evento de efeito único; Único evento alterado; Único evento transiente.

1 Introduction

Satellite communications are a result of research at communication and space technologies, with the main target to achieve different orbits and reducing costs (Maral and Bousquet, 2011). However, big satellites demand heavy rockets and a significant amount of money for their launching. It means that the development of space technologies always was restricted to a few countries, and the commercial exploration still yet a privilege for a minority (Casaril et al., 2018).

In this context, the development of nanosatellites has been growing in the space communication market. Nanosatellites have lower technology complexity, and their launching cost is significantly smallest than regular satellites. Their small size, low cost, and short-time development allow them to be widely selected in Scientific Applications, Signal Monitoring, or Earth observation, for instance (de Carvalho et al., 2012).

Nanosatellites usually are composed by a mechanical structure, an On-Board Computer (OBC – the nanosatellite's brain), an Energy Power System (EPS) to control the production and energy storage system (solar panels, battery, management card, etc.), and a Telecommunications System (antennae, radio, etc.). One way to reduce the raw material cost of a nanosatellite is the adoption of Commercial Off-The-Shelf components (COTS) to build its electronic control boards. Another way is to reduce the number of boards, and hence the number of electronic devices. It may be achieved using the System-on-Chip (SoC) concept that may be applied using a Field Programmable Gate Array (FPGA).

A critical issue related to the adoption of COTS is that electronic devices are exposed to harsh radiation environments and can suffer from hardware faults due to cosmic radiations. These faults can be temporary or permanent. Temporary faults may be classified as Single Event Upset (SEU) and Single Event Transient (SET). A SEU happens every time an energetic particle directly hits a storage element (e.g., memory cell, register, latch, and flip-flop) causing enough charge disturbance to modify the stored value (Baumann, 2005). In contrast with SEUs, which have an error rate independent of the circuit clock frequency, a SET may only generate a soft error if the transient pulse arrives at the input of the memory element during the latching edge of the clock.

To be considered reliable, the nanosatellite' structure must apply fault-tolerant techniques to keep its electronics systems working. For multi-core systems, the reliability is achieved by applying those techniques to the processing elements and interconnection architecture. An alternative is to protect the device with redundancy, i.e., extra functionalities with the sole purpose of detecting and correcting errors, that would not be necessary in a fault-free environment. In the case of FPGAs, redundancy-based strategies are often preferred, as they can be used with COTS components. This way, not requiring the development and fabrication of custom devices, consequently providing a lower cost and better time to market.

This paper introduces the evaluation of a Network-on-Chip (NoC) that connects up to four soft-core processors, forming an SoC. The Network-on-Chip was implemented using two techniques: Hamming code and TRM. Our focus was to evaluate these techniques, checking the silicon consumption and the capability to recover from temporary faults. The protection against transient faults was confirmed by simulations carried out with a software script, done for the ModelSim simulator, that allows the injection of errors in different parts of the network. When compared to the triple modular redundancy (TMR), the Hamming code strategy proved to be efficient. It provides similar levels of fault tolerance with a much lower silicon area cost.

This document was organized as follow: Section 2 introduces the related work about fault-tolerant techniques; Section 3 describes the network proposed, in terms of topology, packet format, physical channels, and adopted mechanisms to deal with faults; Section 4 introduces experimental results related to the network latency, and Section 5 introduces the silicon consumption for the proposed network; Section 6 presents the outcome related to faults simulated with the ModelSim tool. Section 7 finish the paper outlining the main features of the proposed network.

2 Related work

Redundancy-based strategies are very flexible. Implementations can be made at different levels of the circuit design flow, with varying degrees of protection. Most of the developed strategies found in literature fall into four main categories: hardware redundancy, information redundancy, time redundancy, and software redundancy (Goloubeva et al., 2006). This section reports studies that cover some of these categories.

Samudrala et al. (2004) have proposed a technique for hardening combinational circuits mapped onto an FPGA from Xilinx manufacturer (Virtex' family). Their focus was to develop a solution for SEUs. The strategy, called "Selective Triple Modular Redundancy" (STMR), was based on the traditional TMR approach. However, to obtain reduced silicon area overhead, they selectively employ the redundancy in only the most sensitive circuits. The identification of the most sensitive circuits was made by analyzing the signal probabilities of each logic gate within the digital circuit. According to the authors, the experimental results showed that the technique provided immunity against SEUs comparable to the full TMR when used along with other mitigation features of the Virtex FPGA. The area overhead of the STMR strategy may reach nearly 70% of results giving by TMR.

Pratt et al. (2006) also explore the idea of protecting only the most critical sections of a design. They mitigated the effects of SEUs in the configuration memory of the FPGA, instead of the combinational

logic. The authors labeled the configuration bits, used by design mapped onto the FPGA, as sensitive bits and suggested that the sensitive bits could be split into two categories: persistent and non-persistent. A non-persistent configuration bit is a sensitive configuration bit that may introduce functional errors when upset by radiation. However, it can be repaired with configuration scrubbing, and the functional errors disappear. On the other hand, a persistent configuration bit is a sensitive configuration bit which, even after configuration scrubbing, can not repair the introduced functional errors. The authors have developed a software tool, called BYU-LANL Partial TMR (BLTmr), that classifies the circuit structures. According to those authors, the experimental results showed that for a specific design the number of faults in the configuration bits that led to non-repairable functional errors were reduced in two orders of magnitude with a hardware cost of 40% over the unmitigated design.

Gomes et al. (2015) explore the concept of approximate logic circuits to reduce the silicon area overhead of the TMR technique. The idea consists of using approximate logic modules to compose the redundant modules of the TMR. The approximate circuits were modified versions of the original circuit with a smaller silicon area, and also differs its output from the original circuit for a small set of input vectors. Nevertheless, this technique imposes a condition on the approximate circuits: only one of the modules can be different from the original circuit at each input vector scenario, therefore allowing the majority voters to still select two-match outputs out of three for any input vector. According to the authors, the experimental results showed that for a 4-bit ripple carry adder the fault coverage could go up to 93% with 136% of silicon area overhead and 96% with 168% of silicon area overhead. The circuits are mapped using the ABC logic synthesis tool and an academic cell library.

Sanchez-Clemente et al. (2015) have presented another study that also explores the approximate logic approach. However, the technique developed by them aimed at FPGA implementations. In this sense, the valuable logic approximations are the ones that reduce the number of Lookup Tables (LUTs) in the circuit, either by eliminating LUTs or by merging contiguous LUTs. Otherwise, the result is a degradation of the logic function of the circuit. It means that there are no benefits in terms of resource utilization.

Feng et al. (2010) have developed a software-based approach, called Shoestring. That software provides probabilistic soft error reliability. The purpose of the technique was to present high soft error coverage with very little overhead. To achieve these goals, the authors proposed a strategy that combines low-weight symptom-based fault detection schemes with software-based instruction duplication. Symptom-based detection schemes recognize that applications often exhibit abnormal behavior in the presence of soft errors. Although symptom-based detection is inexpensive, it has a limited fault coverage, requiring the use of other techniques concurrently. The main contribution of the Shoestring technique is to efficiently select between relying on symptoms or applying instruction duplication to each part of the program code. The selection analysis runs at compile-time, by introducing new reliability-aware code generation passes into the standard compiler flow. According to the authors, the technique achieved an overall user-visible failure rate of 1.6%, with a performance overhead of 15.8%.

Rebaudengo et al. (2003) have introduced a study case that analyzes the effects of SEUs in the soft processor called LEON. The authors have compared two alternative fault injection techniques: a software-based approach and an emulation-based approach. For the emulation-based approach, the authors have used an FPGA-based platform. For running the injection experiments, a host computer was used as a Fault Injection Manager and communicated with the FPGA board. The authors state that the emulation technique had an accuracy much higher than the software one (up to 13 times higher), concluding that the software approach may lead to significant errors during the error rate estimation.

Touloupis et al. (2007) also introduced the results over a soft processor, called LEON2. In that paper, the authors adopted a simulation-based fault injection approach. In the technique, the primary fault injection support is implemented through a non-synthesizable VHDL entity that has access to all registers and can alter their contents at specified times. The entire system is simulated in a commercial VHDL simulator (ModelSim). They have used the simulator's Foreign Language Interface (FLI) to implement various entities that deal with the fault injection, monitoring, and data collection. In the results, the authors introduced a detailed analysis of the fault effects in the LEON2 processor, particularly in the pipeline unit. According to them, there was an inefficacy of the LEON2 exception mechanism for detecting injected faults, as well as the dependence between the observed fault effects and the processor's workload.

Travessini et al. (2018) analyzed a fault injection campaign in some registers of the LEON3 soft-processor. They injected the faults using simulation scripts that force a bit flip while the processor is running a set of three different workloads. The script was written for the ModelSim simulator, and the study was restricted to single bit upsets (SBU). They investigated the effects of these injected faults and how they propagate to the CPU core boundaries. According to the authors, the obtained results showed that the majority of the failures were due to faults injected in only a small number of the processor registers. Furthermore, in that study, it was proposed a partial triple modular redundancy approach to protecting only the CPU's most sensitive registers, achieving a 99.25% SBU tolerance with a marginal increase in area.

Villa et al. (2018) introduced a fault-tolerance technique, based on the concept of temporal redundancy, with checkpoints and recovery for soft-core processors. They also worked with LEON3 soft-processor. The proposed modified architecture was

aimed at embedded systems for spatial applications, based on FPGAs. According to them, the experimental results showed that the Checkpoint and Recovery (CR) technique was a valid alternative to TMR and even Dual Modular Redundancy (DMR). These results are essential when considering the limited logic area and power budget present on a satellite. The results have comparable levels of reliability to the more conventional fault-tolerance techniques. They state that their approach does not require modifications to the software source code or compiler.

Fuchs et al. (2019) proposed solution for nanosatellites. They developed a software-fault-tolerance-approach based upon thread-level coarse-grain lockstep, which was validated using fault-injection. To offer strong long-term fault coverage, their architecture was implemented as tiled MPSoC on an FPGA, utilizing partial reconfiguration, as well as mixed critically. According to them, that architecture can satisfy the high performance requirements of current and future scientific and commercial space missions at very low cost, while offering the strong fault-coverage guarantees necessary for platform control even for missions with a long duration.

2.1 Conclusion about related work

In this section, we introduced some of the redundancy-based techniques to deal with temporary faults. Most of them work with hardware, software, soft-processor, and temporal redundancy. So far, we did not found papers about Network-on-Chip, applied to nanosatellites, and with protection against temporary errors caused by spacial radiation. The only reference is the paper proposed by Fuchs et al. (2019). However, their solution was based on software and partial reconfiguration. We understand that our proposal fulfills this gap because it does not involve partial reconfiguration nor software solutions.

3 Network proposal

The Network-on-Chip proposed in this document was designed to be a communication fabric capable of interconnecting up to four soft-processors, and to build a *System-on-Chip* for nanosatellites. For the development, we based on the following assumptions:

1) the network must support up to four soft-processors;
2) the processors must have their own mechanisms to deal with faults due to cosmic radiation;
3) the FPGA configuration memory must be updated by an protection mechanism and external of FPGA; and
4) the silicon consumption must be minimized to reduce the probability of fault occurrences.

The first assumption is related to the reference nanosatellite. The SoC design will replace four boards that control the Floripasat[1] nanosatellite. It is a CubeSat 1U with 10 cm^3. Each board of Floripasat has a microcontroller: OBDH (On-Board Data Handling), TT&C (Telemetry, Tracking, and Command), EPS (Electric Power System), and Payload. These boards are depicted in Fig. 1, and they are interconnected by industrial standard connectors called PC-104[2].

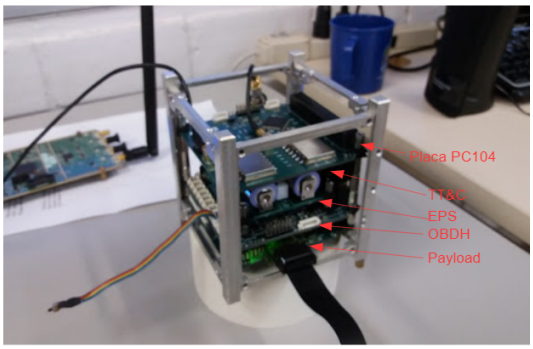

Figure 1: Image of FloripaSat internal boards.
Source: Available at https://floripasat.ufsc.br

The OBDH acts as the Central Processing Unit. The TT&C board has radio-frequency modules and is responsible for the communication between the nanosatellite and the ground station. The EPS board controls the charge of the batteries and the photo-voltaic panels that collect the energy from the sun. The Payload board is responsible for dealing with experiments that are running in the spacial environment.

The second and third assumptions are essential to establish the boundaries of this work. A periodic reconfiguration on FPGA and the adoption of soft-processors with protection against cosmic radiation will ensure that the protection proposed for the NoC will be enough for the correct working of the nanosatellite.

Finally, the fourth assumption was based on the logic that fewer silicon areas naturally will imply the lower probability of fault occurrences. Based on this, there are four routers in the network, and the data flows only though clockwise from one router to another one.

3.1 Topology

Based on the first assumption stated at the beginning of Section 3, the Network-on-Chip must connect four processing elements. Our choice was a direct ring network in which each processor is connected in a router. The routers will be responsible for sending and receiving data among processors. The deterministic routing was adopted, in which the path is defined according to the origin and destination of each message, and uni-cast because all messages have only one

[1]Available at https://floripasat.ufsc.br.
[2]Available at https://pc104.org/hardware-specifications/pc104/

destination router. Fig. 2 depicts the NoC composed by four routers (R0, R1, R2, and R3), and four processing elements (PE).

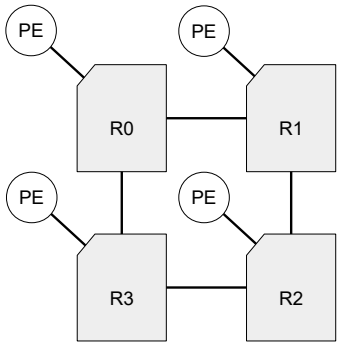

Figure 2: Proposed Network with four router and processors.

3.2 Packet format

A packet is a set of "flits" (flit is acronym of Flow Control Unit). Flit is the smaller data unit in which the flow control is applied. Each flit has 32 bits, of which the two most significant bits indicate if the flit is the beginning of the packet (header), the payload of the packet, or the last flit of the packet (tail). The third and fourth bits indicate the destination router for the packet. The remaining bits are the data, in itself. Table 1 introduces some examples of flits from a packet, and Fig. 3 shows the position of each bit in the flit.

Table 1: Definition for the most significant bits.

	1º and 2º most significant bits	3º and 4º most significant bits
00	*Payload*	Router 0
01	*Payload*	Router 1
10	*Tail*	Router 2
11	*Header*	Router 3

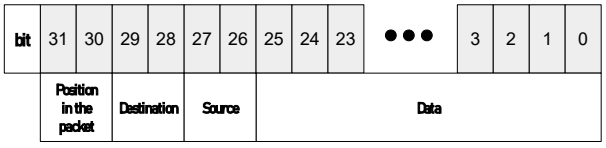

Figure 3: The relation between bits and flits.

The routing of packets in the proposed network is similar to a "token-ring"[3]. There is a token continuously be sending through the network. This token was defined as $7FFFFFFE_h$, and when the power goes up, router 0 starts the communication. It has two options: send a valid packet to router 1 or pass the token to the router 1 if it does not need to transmit valid data. The same situation happens to router 1: it can send valid data or send the token to router 2 (and so on). The token is checked; meanwhile, there is no valid packet been transmitted. Token sent after header flit and before tail flit are considered valid data.

3.3 Physical channels and Routers

We evaluated two versions of Network-on-Chip. The first one adopts the TMR (acronym of Triple Modular Redundancy) technique, meanwhile, another one employees a Hamming Code technique. The silicon consumption change, for each one of these implementations. Details about the silicon consumption will be in Section 4. This section begins explaining the basic structure of the routers, and following introducing the overhead that each approach demand.

3.3.1 *Router for TMR approach*

Every Processing Element (PE) depicted in Fig. 2 inject data that will be sent through the routers up to the destination PE. The internal structure of the routers was designed to be simple; that is, fewer electronics circuits result in a lower faults probability. The internal structure of the router is shown in Fig. 4. This router is adopted for the network builds with the TMR technique.

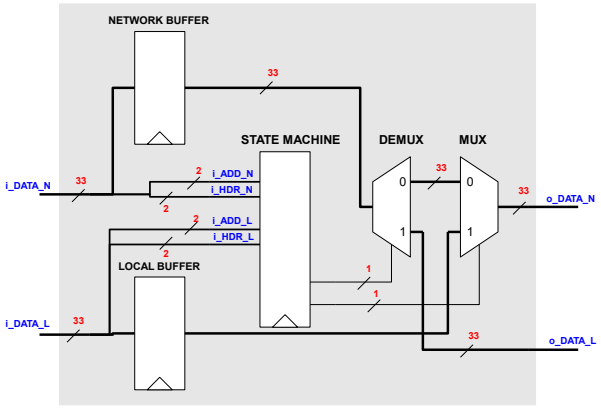

Figure 4: Internal structure of a router.

Data coming from the network gets into the router by the channel **i_DATA_N** (Fig. 4). The state machine checks if bits 33 and 32 are "11". If yes, it checks the bits 31 and 30 to identify the flit's destination (**o_DATA_N** or **o_DATA_L**) and select the proper output of the DEMUX and MUX. A flit from PE connected in the

[3]Token Ring local area network (LAN) technology is a communications protocol for local area networks. This token passing is a channel access method providing fair access for all Processing Elements, and eliminating the collisions of contention-based access methods.

local channel **i_DATA_L** will be sent to the network selecting "1" for the MUX.

3.3.2 *Router for Hamming approach*

The router designed for the Network-on-Chip based on Hamming code is different from the one introduced in Section 3.3.1. For the Network with Hamming code we implemented an Encoder and a Decoder inside the router. Fig. 5 depicts the differences between both routers, in which the Hamming Encoder and Decoder are squares colored by dark Grey color.

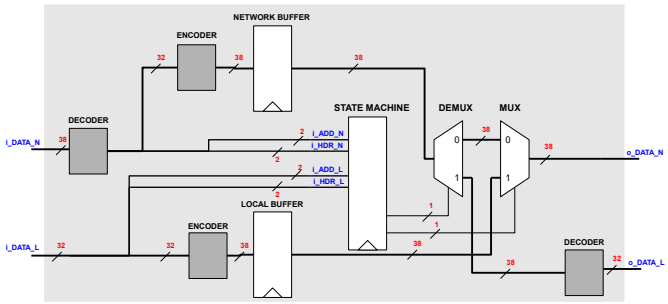

Figure 5: Internal structure of the router with Hamming encoder and decoder.

Hamming codes can detect up to two-bit errors or correct one-bit errors without the detection of uncorrected errors. It adopts extra bits in it, and each flit is transformed into a code-word, which has its length (total bits of coded flit) established by the Eq. (1):

$$N = m + x \tag{1}$$

in which m is the number of bits of data flit and x a fixed amount of control bits. For our implementation, the data traffic flows through the network with 38 bits. It happens because 6 extra bits must be inserted to the original flit, according to Hamming code rules for 32 data bits (Tomlinson et al., 2017). The decoders check this extra bits to verify if there were some error in the data received, and extract the extra bits from Hamming code.

3.3.3 *Triple Modular Redundancy - TMR*

TMR is a passive hardware redundancy, and the redundant elements are used to "mask" the faults. It means that all three elements perform a process, and the result is processed by a majority-voting system to produce a single output. If anyone of the three systems fails, the other two can correct and mask the fault (Goloubeva et al., 2006). Fig. 6 shows a generic example of TMR.

It is essential to take into account that redundancy increases the number of components in the system. The more it increases, the more the fault probability (Weber, 2003). Fig. 7 depicts a voter implementation using registers and logic gates. We are assuming that the combinational circuit is triplicated, and each one

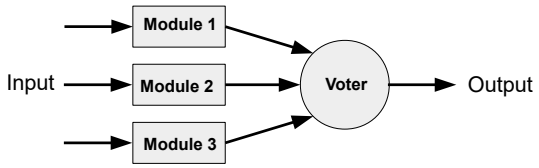

Figure 6: Voter circuit in TMR implementation.
Source: Adapted from Goloubeva et al. (2006).

of its outputs is connected to a register (flip-flop), vulnerable elements for SEU in the voter.

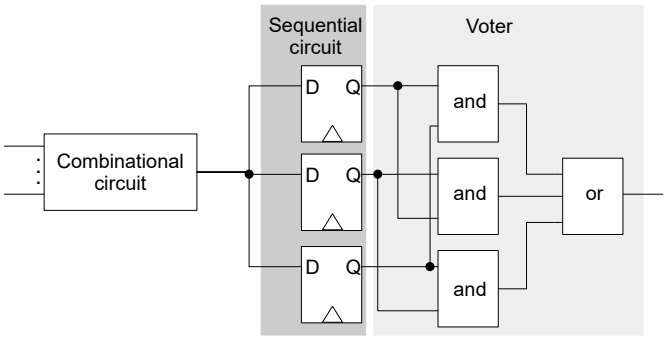

Figure 7: "Voter" circuit - Internal implementation.
Source: Adapted from Kastensmidt et al. (2006)

TMR implementation could be done in two different ways. The first one would be triplicate each internal components of routers. The second one would be triplicate the number of routers because if an internal component of one of the routers had SEE, the other two routers would vote correctly. The first approach would expend more silicon than the second one. Based on this, we decided to implement the second approach, as shown in Fig. 8.

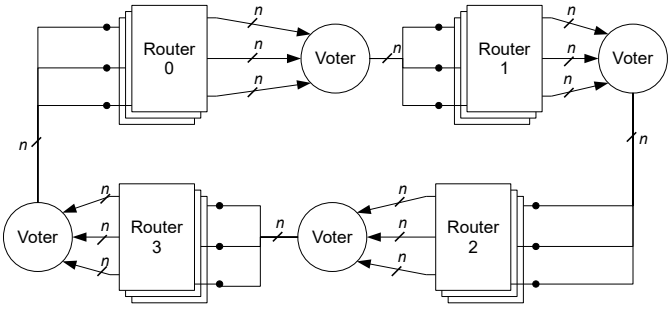

Figure 8: NoC with TMR technique.

Section 4 will introduce the results related to silicon consumption for this implementation.

3.3.4 *Hamming code*

In this approach, the network was built with four routers, which is entirely different from the TMR

approach in which twelve routers were used, further the voters' circuits. Here we added a Hamming Encoder at the output bus of each router. The channels were connected at the input of the subsequent router that has an external Hamming Decoder, as shown in Fig. 9. The amount of signal in each physical channel is given by the expression: $(n + m)$, in which $n = 32$ is the amount of data bits and $m = 6$ are the bits related to Hamming coding.

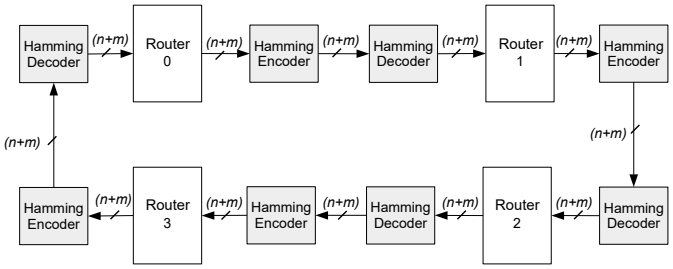

Figure 9: Network-on-Chip with Hamming technique.

4 Experimental Results

This section introduces the experimental results in terms of latency, silicon consumption, and fault tolerance. To do the experiments we used a FPGA from Intel Manufacturer, called EP4CGX22CF19C6, Cyclone IV family. The fault tolerance was done with a script running in Modelsim software tool. Next subsections will introduce these results.

4.1 Evaluation of traffic and network latency

Every flit sent from a router into the network takes four clock cycles to reach the next router. It means that a flit sent from Router 0 to Router 3 will takes twelve clock cycles. Fig. 10 shows a sequence of flits sent from Router 0 (**i_DAT_L0**) up to Router 2 (**o_DATA_L2**). As expected, it takes eight clock cycles due to their positions in the network (Fig. 2).

Several controls in a nanosatellite demand the knowledge of response time because they are considered critical for the nanosatellite operations. Thus, it is essential to know the worst-case latency (WCL) for all communications through the network that will interconnect the soft-processors. For this reason, we decided to establish some assumptions to guarantee the WCL for all flows in the Network-on-Chip we are proposing:

- the maxim packet size is 42 flits, with one header flit, one tail flit and 40 payload flits; and
- every router can send only one packet when it receives the token, with a maximum of 42 flits per packet. After sending the packet, it must send the token to the adjacent router.

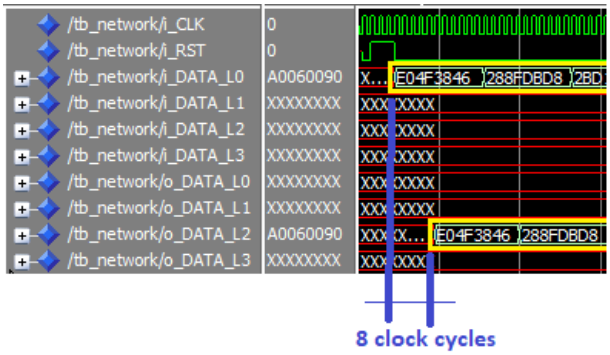

Figure 10: One packet was sent from router 0 (R0) to router 2 (R2).

Let's call T_{clock} the system clock period, and T_{router} the time required by a router to transmit a flit in the network. As mentioned earlier, a router demands four clock cycles to rotate a flit. Thus, the T_{router} can be written as Eq. (2):

$$T_{router} = 4 \times T_{clock} \tag{2}$$

The latency to send a packet from a router to another one is giving by the following Eq. (3):

$$L = (N_{routers} - 1) \times T_{router} \times P_{size} \tag{3}$$

in which $N_{routers}$ is the number of routers involved in communication, and P_{size} is the number of flits in the packet. For example, consider the Processing Elements (PE) connected in the router depicted in Fig. 2. If the PE connected in Router 0 need to send a packet with 16 flits to the PE connected in Router 2, the latency of this packet is calculated as Eq. (4):

$$L_{PE_0} = (3 - 1) \times T_{router} \times 16 = 32 \times T_{router} = 128.T_{clock} \tag{4}$$

So, this packet will take 128 clock cycles to be transmitted from Router 0 up to Router 2. We can use the Eq. (3) to predict the worst case latency (WCL) for a packet in the proposed NoC. The WCL will happen when the Router 0 must send a packet with 42 flits (maximum allowed for each transfer according our assumptions) to Router 3. The expression for WCL is given as Eq. (5):

$$L_{WCL} = (4 - 1) \times T_{router} \times 42 = 504.T_{clock} \tag{5}$$

We are assuming that the latency inside each PE to process the packet is well known, and is out of our scope.

5 Silicon consumption

We used the reports from the Quartus II tool, from Intel manufacturer, to check the silicon consumption for three different implementations (without fault-tolerant technique, using TMR and using Hamming). These reports introduce silicon consumption in terms of Logic Elements (LE) and registers. Both networks were implemented in the FPGA, part number EP4CGX22CF19C6. A Logic Element is composed of the following electronics components: a memory called LUT (acronym of Look Up Table), some flip-flops, some multiplexers, and some combinational logic. Fig. 11 depicts a block diagram with these electronics components.

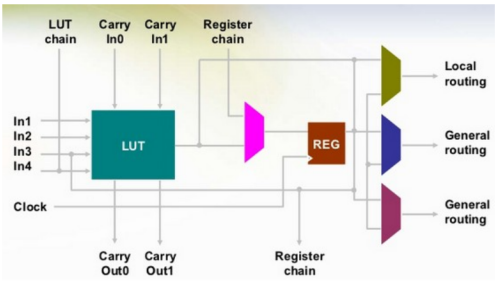

Figure 11: The diagram shows a Logic Element for the FPGA family called Cyclone IV, from Intel manufacturer.
Source: Available at: https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-iv.html

The total of Logic Elements available in EP4CGX22CF19C6 FPGA is 109,424 LE. The network without fault-tolerant technique consumed 1,032 LE, which is less than 1% of the total LE available. Table 2 shows the silicon consumption for the three versions of the NoC: a Noc without fault-tolerant technique, a Noc with TMR, and an NoC with Hamming code technique. Note that the silicon consumption for the NoC with Hamming code almost doubled (99%), when compared with the regular NoC. Even so that consumption is much lower than the version implemented with the TMR technique (233%). Fig. 12 depicts graphically these silicon consumption.

Table 2: Silicon consumption for three versions of networks: regular, with TMR, and with Hamming code.

NoC	Logic Elements (LE)	Registers	% (LE)
NoC	1.032	602	-
NoC TMR	3.437	2.005	233
NoC Hamming	2.050	942	99

Another silicon consumption evaluation was done, taking into account the internal components of a single Router, and including the Hamming encoder and decoder. Table 3 shows the silicon consumption for this case. Note that the encoder and the decoder have higher silicon consumption than the internal components of the router. Fig. 13 depicts graphically these silicon consumption. The Hamming encoder was implemented with only combinational logic, so the consumption of registers for this encoder was zero.

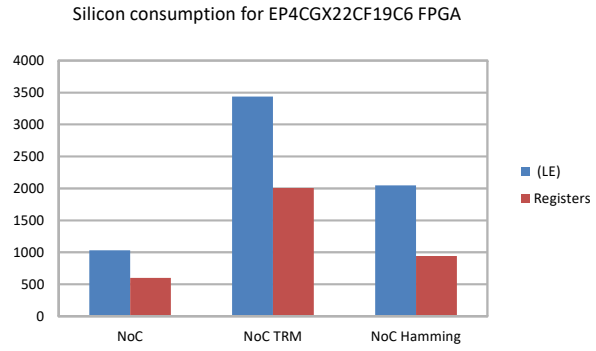

Figure 12: Silicon consumption for the NoCs: regular, with TMR technique, and with Hamming code.

Table 3: Silicon consumption for internal components of a Router and Hamming modules.

Components	Logic Elements (LE)	Registers
Router	456	160
State machine	60	19
Encoder	84	0
Decoder	882	310

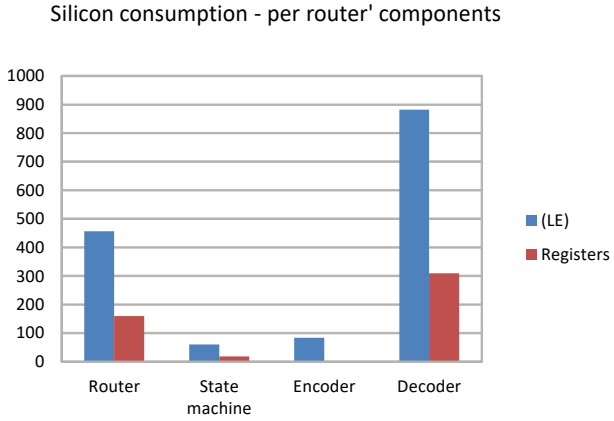

Figure 13: Silicon consumption for internal components of Router and Hamming coding.

6 Fault simulation with ModelSim

There are two ways to verify if the fault-tolerant mechanisms we are evaluating work well for temporary faults due to cosmic radiation. The first one is: to submits an FPGA to receive direct radiation at a specialized laboratory, such as Peletron[4] (USP – University of São Paulo – Brasil). Another way is using a specific tool capable of simulating those faults.

We decided to adopt a technique to inject random faults in the FPGA using the ModelSim software tool. We used a script written in VHDL language, similar to the work did by Travessini (2018). We let the network always sending data. Flits were sent from Router 1 to Router 3, and from Router 2 to Router 0.

Our focus was on generating one fault and verify the regular NoC (without fault-tolerant mechanism), the TMR, and the Hamming. It was expected that TMR and Hamming NoCs were able to identify and correct the error injected in one of the routers. Each test consists of generating one fault, inverting the value of one logical bit, randomly. These fault injections were done for all routers connected in the network. Fig. 14 shows the script we used in fault simulations.

```
# Arguments:
#  root         root signal that will be traversed, it can be a vhdl record, a
#               vector, or only a bit.
#  listName     the list variable where the result signals will be written
proc splitSignal {root listName} {
    upvar $listName list
    set signalValue [examine $root]
    if {[string length $signalValue] == 1} {
        # The the signal is only one bit.. add to list
        lappend list $root
    } elseif {[string match *\{* $signalValue]} {
        # The signal may be a record or an array
        if {[llength [find signals -internal -r ${root}.*]]} {
            # Record
            set children ${root}.*
        } elseif {[llength [find signals -internal -r ${root}(*)]]} {
            # Array
            set children ${root}(*)
        } else {
            error "Invalid signal type"
        }
        # Traverse the record/array
        foreach child [find signals -internal -r $children] {
            splitSignal $child list
        }
    } else {
        # The signal is a vector, traverse through each bit
        set bits ${root}(*)
        foreach bit [find signals -internal -r $bits] {
            splitSignal $bit list
        }
    }
    return
}
```

Figure 14: TCL Procedure that traverses a root signal and lists all the internal signals bit-wise.
Source: adapted from Travessini (2018).

Next subsection, we will discuss more the Network with Hamming encoding because it is the solution with lower silicon consumption, which is a crucial issue for nanosatellites.

[4]Access at http://portal.if.usp.br/fnc/pt-br/acelerador-pelletron

6.1 Fault simulation in NoC with Hamming encoding

We are assuming one temporary fault at a time. A Fault was injected at each simulation, at random routers, inside the packet transmission period. As expected, for all simulations, the right packet arrives at its destination router, and without any error.

Let us check one of these experiments. The original bit was 0, and it was changed to 1 in a flit. Note that, even the noise happening and changing the value of eighth bit, changing the flit from "0A11FBBD43" into "0A11FBBDC3" (see Table 4, the changing bits are at orange color), the network was able to delivery to the destination the same packet coming from the Router 1.

Table 4: Fault injection at eighth bit.

Hexadecimal packet	Binary packet
0A11FBBD43	00 1010 0001 0001 1111 1011 1011 1101 0100 0011
0A11FBBDC3	00 1010 0001 0001 1111 1011 1011 1101 1100 0011

Thus, considering the Fig. 4, if an SEU strikes one of the internal storage components of the router (Network Buffer or Local Buffer), is possible to note that the destination address (*i_ADDR*) and the signal related to the beginning and end of packets (*i_HDR*) will be sent correctly to the state machine. It happens because they do not come from the buffers. The flit will be forwarded through the network with a error in one of its bits. When this flit achieves the destination router, first, it passes through the Hamming decoder that will find the error, and it will be revised.

If the cosmic radiation strikes a Bit in combinational logic, what configures a SET will be more difficult to generate a soft error because its necessary that the transient pulse reaches the input of the memory element during the rising edge of the clock. Even though, when the data pass through the Hamming decoder (at the local output, or the global input), the error will be noted and fixed. Furthermore, the probability of a temporary fault that happens twice in the same flit is extremely low; we understand that only the treatment done by this mechanism is enough to ensure reliability for these categories of failure (temporary).

7 Conclusion

Our work focused on the development of a Network-on-Chip (NoC) capable of interconnecting up to four processing elements. The NoC is fault-tolerant against cosmic radiation, for nanosatellite at low Earth orbit (LEO).

The NoC's design was the most simple as possible because a small silicon area implies the lower probability of cosmic radiation strike. It has a 2D topology, and the packets flows are sending in clockwise. The adoption of a token-ring mechanism

with a limitation for the packet size and one packet at a time allows knowing the worst-case latency for all communication through the Network. We evaluated two different mechanisms to protect the Network against temporary faults: a TMR and Hamming code. We did an extensive fault injection campaign to confirm the efficiency of both mechanisms. The fault injection was done using a script wrote to change the status of a Bit in the network, random. The software tool used to do these simulations was the ModelSim, from the Mentor company.

Our focus was on temporary faults because the majority of nanosatellites do not have propulsion engines. They usually are released at LEO using a rocket and keep on that orbit for two years before getting into the atmosphere and burn. We understand that the NoC with Hamming coding proposed in this document is efficient for nanosatellites that work on this context, and we believe that this is the first Network-on-Chip conceived for nanosatellites.

References

Baumann, R. C. (2005). Radiation-induced soft errors in advanced semiconductor technologies, *IEEE Transactions on Device and Materials Reliability* **5**(3): 305–316. https://doi.org/10.1109/TDMR.2005.853449.

Casaril, L., de Souza, B. L. and Brito, A. G. (2018). Simulação orbital de nanosatélites, *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics* **6**(1). Available at https://proceedings.sbmac.org.br/sbmac/article/view/1751.

de Carvalho, M. J. M., Jotha, L. S., Lima, J. S. S., Biondi, R. B., Aquino, P. S. and Lima, D. N. A. (2012). Estudo de estratégias de mitigação de detritos espaciais para uma constelação de nano satélites de coleta de dados ambientais, *Proceedings of 16th ISU Anual International Symposium*, INPE, Strasbourg, FRA.

Feng, S., Gupta, S., Ansari, A. and Mahlke, S. (2010). Shoestring: Probabilistic soft error reliability on the cheap, *SIGPLAN Not.* **45**(3): 385–396. http://doi.acm.org/10.1145/1735971.1736063.

Fuchs, C. M., Murillo, N., Plaat, A., der Kouwe, E. V., Harsono, D. and Stefanov, T. (2019). Fault-tolerant nanosatellite computing on a budget, *Conference on Radiation Effects on Components and Systems (RADECS)* **1**(1). Available at https://openaccess.leidenuniv.nl/handle/1887/83305.

Goloubeva, O., Rebaudengo, M., Reorda, M. S. and Violante, M. (2006). *Software-implemented hardware fault tolerance*, Springer Science & Business Media, Torino, ITA.

Gomes, I. A., Martins, M. G., Reis, A. I. and Kastensmidt, F. L. (2015). Exploring the use of approximate tmr to mask transient faults in logic with low area overhead, *Microelectronics Reliability* **55**(9): 2072 – 2076. Proceedings of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

Kastensmidt, F. L., Carro, L. and da Luz Reis, R. A. (2006). *Fault-tolerance techniques for SRAM-based FPGAs*, Vol. 1, Springer, Dordrecht, NED.

Maral, G. and Bousquet, M. (2011). *Satellite communications systems: Systems, Techniques and Technology*, John Wiley & Sons, Hoboken, USA.

Pratt, B., Caffrey, M., Graham, P., Morgan, K. and Wirthlin, M. (2006). Improving fpga design robustness with partial tmr, *2006 IEEE International Reliability Physics Symposium Proceedings*, pp. 226–232.

Rebaudengo, M., Reorda, M. S. and Violante, M. (2003). Accurate analysis of single event upsets in a pipelined microprocessor, *Journal of Electronic Testing* **19**(5): 577–584. https://doi.org/10.1023/A:1025130131636.

Samudrala, P. K., Ramos, J. and Katkoori, S. (2004). Selective triple modular redundancy (stmr) based single-event upset (seu) tolerant synthesis for fpgas, *IEEE Transactions on Nuclear Science* **51**(5): 2957–2969. https://doi.org/10.1109/TNS.2004.834955.

Sanchez-Clemente, A., Entrena, L. and Garcia-Valderas, M. (2015). Partial tmr in fpgas using approximate logic circuits, *2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pp. 1–4.

Tomlinson, M., Tjhai, C. J., Ambroze, M. A., Ahmed, M. and Jibril, M. (2017). *Error-Correction Coding and Decoding*, Springer. https://doi.org/10.1007/978-3-319-51103-0.

Touloupis, E., Flint, Member, J. A., Chouliaras, V. A. and Ward, D. D. (2007). Study of the effects of seu-induced faults on a pipeline protected microprocessor, *IEEE Transactions on Computers* **56**(12): 1585–1596. https://doi.org/10.1109/TC.2007.70766.

Travessini, R. (2018). Low overhead soft error reliability improvement for soft processors, Dissertação (Mestrado) — Universidade Federal de Santa Catarina (UFSC).

Travessini, R., Villa, P. R. C., Vargas, F. L. and Bezerra, E. A. (2018). Processor core profiling for seu effect analysis, *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pp. 1–6. https://doi.org/10.1109/LATW.2018.8347235.

Villa, P. R. C., Travessini, R., Vargas, F. L. and Bezerra, E. A. (2018). Processor checkpoint recovery for transient faults in critical applications, *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pp. 1–6. https://doi.org/10.1109/LATW.2018.8349674.

Weber, T. S. (2003). Tolerância a falhas: conceitos e exemplos.