2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)

Optimized Fault-Tolerant Buffer Design for Network-on-Chip Applications

Alan C. Pinheiro, Jarbas A. N. Silveira, Daniel A. B. Tavares, Felipe G. A. Silva, and César A. M. Marcon

***Abstract*—Newest technologies of integrated circuits manufacture allow billions of transistors arranged in a single chip, which requires a communication architecture with high scalability and parallelism degree, such as a Network-on-Chip (NoC). As the technology scales down, the probability of Multiple Cell Upsets (MCUs) increases, being Error Correction Code (ECC) the most used technique to protect stored information against MCUs. NoC buffers are components that suffer from MCUs induced by diverse sources, such as radiation and electromagnetic interference. Thereby, applying ECCs in NoC buffers may come as a solution for reliability issues, although increasing the design cost and requiring a buffer with higher storage capacity. This paper proposes an optimized buffer using an Extended Hamming code to deal with MCUs and enhance the protected information storage, pursuing to reduce the area and power required for ECC implementation. We guide the optimized buffer evaluation by measuring the fault tolerance efficiency, buffer area, power overhead and performance of the proposed technique. All tests included the comparison with a non-optimal appliance of ECC in a NoC buffer. The results show the proposed technique reduces the area and power overhead in buffers with ECC and allows a considerable fault tolerance against MCUs with a small performance impact.**

***Index Terms*—Network-on-Chip, Fault Tolerance, Buffer Optimization, Error Correcting Code (ECC)**

I. Introduction

THE increasing development of VLSI technologies allowed integrating multiple logic cores in a single System-on-Chip (SoC), which provides integrated solutions for countless applications in several areas, such as entertainment, telecommunication and consumer electronics [1]. These systems require high scalability, massive parallel communication and, sometimes, stringent timing constraints. Network-on-Chip (NoC) is an efficient packet-oriented communication structure [2], presenting high flexibility, scalability, and parallelism [3], fulfilling the communication requirements of such applications.

Along with the benefits of on-chip communications, the NoC architecture is quite simple, composed of routers, links and Network Interfaces (NIs) to connect Processing Elements (PEs) such as processors and memories. Routers, containing control logic, crossbars and buffers, dispatch the packets sent by PEs across the NoC. Depending on the NoC topology, global links interconnect the routers, and local links connect a router to NI and NI to the local PE [4].

A Multiprocessor SoC (MPSoC) provides a vast processing capacity requiring efficient communication architecture. Therefore, a NoC-based MPSoC is becoming an outstanding solution for critical CPU-bounded systems, which have been widely applied in network security and data transmission, providing significant improvement for Real-Time Systems [5].

A growing problem for the recent SoC manufacturing technologies is the increase of Single Event Effects (SEEs) incidence, causing transient faults. SEEs are induced by many sources, such as radiation, coming from cosmic emissions and electromagnetic interference [6]. Achallah, Othman and Saoud [7] investigate the problems and challenges of the NoC design, like the guarantee of correct communication in the presence of transient and/or permanent faults.

Error Correcting Code (ECC) is the most common solution to mitigate data errors of critical applications [8]. NoC buffers may be affected by the strike of a charged particle inducing information errors, compromising the data transmission. Therefore, systems based on NoC structures in hostile environments for electronic devices must provide some reliability degree to maintain the application working correctly. Radetzki et al. [9] present an overview of fault tolerance techniques applied to NoCs, covering cost and performance tradeoffs of the ECC implementation in input buffers.

Silveira et al. [10] describe the Phoenix MPSoC, which has as a 2-Dimensional NoC as communication architecture. Each link of the NoC implements a Fault Prediction Module (FPM) employing the Extended Hamming (ExHamming) code. The FPM circuit allows correcting or retransmitting faulty flits and ranking links regarding the number of faults during a given window of time, allowing the routing logic of the NoC to find the safest path to deliver packets.

Silva et al. [11] present the utilization of different types of ECCs in NoC buffers. The results showed the higher is the complexity of the ECC applied, the higher is the costs of the buffers.

This paper proposes a fault-tolerant NoC buffer that uses ExHamming and a management structure responsible for optimizing the memory resource use. This approach aims to enhance the protection and cost of the data stored in the buffers, with low-performance impact. The remainder of this paper is

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Alan C. Pinheiro, Jarbas A. N. Silveira, Daniel A. B. Tavares and Felipe G. A. Silva are with the Teleinformatics Engineering Department, Federal University of Ceará, Fortaleza, Brazil (email: cadore@lesc.ufc.br, jarbas@lesc.ufc.br; danieldgt@gmail.com; gaspar@lesc.ufc.br).

César A. M. Marcon is with the Computer Science Department, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil (email: cesar.marcon@pucrs.br).

978-1-7281-0453-9/19/$31.00 ©2019 IEEE

structured as follows. Section II details the materials and methods used in this work. Section III shows the experimental results. Finally, Section IV presents our main conclusions.

II. Materials and Methods

The proposed design was implemented in NoC Phoenix, which has regular mesh topology, wormhole switching and five dual-channel ports per router. Each router uses XY routing algorithm, On-Off flow control, and Round-Robin arbitration. Fig. 1 presents the router structure of NoC Phoenix.

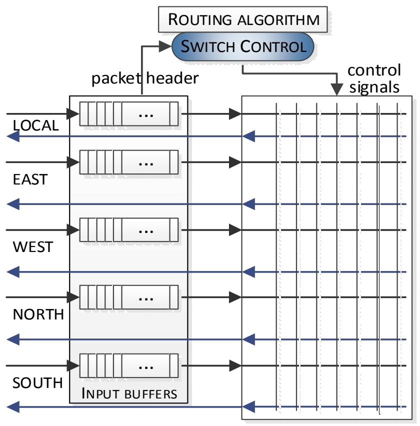

Fig. 1. Router structure of NoC Phoenix.

Every router input port (NORTH, EAST, WEST, SOUTH, and LOCAL) has a buffer structure (*Regular Buffer*), which implements a circular FIFO with 11-flit depth and 16-bit width.

Fault-tolerant circuits are crucial to preserving the correct operation of critical components [12]. Buffers are critical components of a NoC, responsible for storing the data traffic temporarily. As fault-tolerant techniques consume many resources, the employment of ExHamming is an economic approach considering its lower design cost compared with other ECCs [11]. ExHamming is a Single Error Correction-Double Error Detection (SEC-DED) and was developed for a 16-bit data with 6-bit redundancy.

To evaluate the efficacy of our proposal, we implemented two versions of buffers besides the original implementation of the NoC Phoenix: (i) *Extended Buffer* and (ii) *Optimized Buffer*. The original version of NoC Phoenix contains buffers (*Regular Buffer*) without a fault-tolerant mechanism. The *Extended Buffer* version keeps the same buffer depth, but adding a portion of redundancy alongside each flit of 16 bits, becoming a buffer of 11 x 22 bits (i.e., 6 redundancy bits representing the ExHamming syndrome for a 16-bit data). The *Optimized Buffer* version is the proposal for this paper and will be explained next.

To compare with the proposed solution, we replicated the ECC implementation for a NoC buffer presented in [11], which also uses ExHamming (*Extended Buffer)*. Both methods were evaluated according to area consumption and power dissipation; varying the NoC size in 2×2, 3×3 and 4×4, fault-tolerant capability and traffic performance (considering only the 4×4 NoC dimension). Fig. 2 illustrates the *Extended Buffer* version and the data flow with this structure.

The *Extended Buffer* version extends the buffer width from 16 to 22 bits, encoding and storing 11 data flits. This approach encodes each data-flit on every pushing procedure before writing in the buffer. The redundancy bits are written at the same duty cycle as the data flit. When the data pull procedure happens, it is necessary to access only one address in the buffer to decode correctly, also, requiring only one duty cycle.

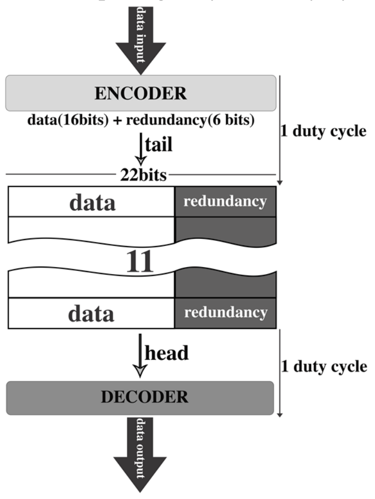

Fig. 2. *Extended Buffer* version.

The *Optimized Buffer* version is an improved solution that reduces the total of storable information, but preserving the *Regular Buffer* structure (width = 16 bits, depth = 11). Fig. 3 shows the *Optimized Buffer* version with 8 addresses for data flit and 3 addresses to store all the redundancies.

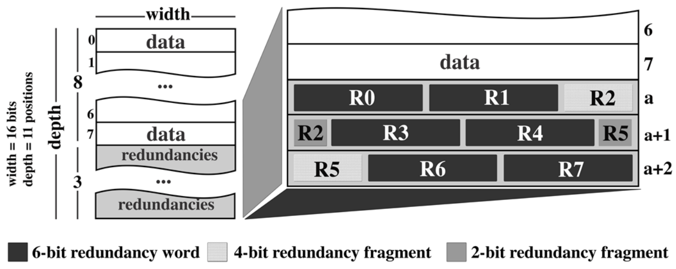

Fig. 3. *Optimized Buffer* version.

The redundancy of each data flit is stored in the redundancy region, following the data address in the buffer; i.e., the data flit of address 0 will have its redundancy R_0 entirely stored in address a. However, some redundancies could be split into 2 flits, to avoid memory fragmentation; i.e., for example, the redundancy related to the address 5 data is stored in R_5, but first 2 bits are stored at address $a + 1$ and the remaining 4 are stored at the address $a + 2$. Table I presents the configurations of the *Optimized Buffer* version to avoid memory segmentation.

TABLE I. NUMBER OF FLITS NECESSARY IN THE BUFFER FOR EACH CONFIGURATION CASE (N = 1, 2, 3 … ∞).

Case	Data	Redundancy	Buffer depth
1	8	3	11
2	16	6	22
3	24	9	33
n	n × 8	n × 3	n × 11

The proposed *Optimized Buffer* version uses the 3-flit redundancy ratio for every 8-flit data. Fig. 4 shows the data flow of the *Optimized Buffer* version.

Each procedure of data pushing or pulling uses only one duty cycle. In the data pushing flow, after encoding, the redundancy bits are written in the redundancy region, filling the three addresses of the 16-bit restricted to redundancy bits. When an address is filled, the next address will receive the redundancy bits, while the data is written in the data region. Whenever the data pulling happens, the data flit and its redundancy bits are read and decoded. Notice that one or more redundancy addresses can be read, as the push proceed.

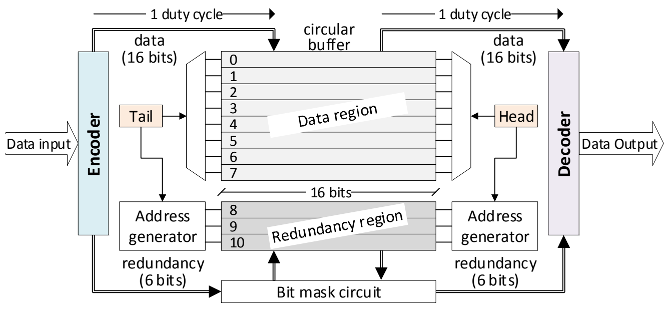

Fig. 4. Data flow of the *Optimized Buffer* version.

The *Optimized Buffer* scheme is expected to provide reliability for the NoC buffer such as the *Extended Buffer*, although the *Optimized Buffer* version has lower resources and power impact than the *Extended Buffer* version. Next section presents the synthesis evaluation for both versions, comparing to the *Regular Buffer* scheme of the NoC Phoenix. Then, an analysis of the error coverage was made comparing *Optimized Buffer* to *Extended Buffer*, followed by their performance analysis through the simulation of uniform synthetic traffic in the NoC.

III. Experimental Results and Discussion

The evaluation of the proposed work was divided into three experiments: (i) Synthesis Cost Impact, (ii) Error Coverage Analysis and (iii) Performance Evaluation. The first one exposes the costs of area consumption and power dissipation, comparing the fault-tolerant buffer versions to the *Regular Buffer*. The second one describes and discusses the results achieved concerning error coverage of the *Optimized Buffer* and *Extended Buffer* schemes, considering that the bits are physically positioned as Fig. 2 and 3. Finally, the last one analyses the performance influence of the real data capacity variation in both fault-tolerant versions.

A. Synthesis Cost Impact

The NoC Phoenix together with the buffer versions were designed in VHDL and synthesized with *GENUS* Cadence's tool for a CMOS 65 nm technology. This experiment employs mesh NoCs with sizes: 2×2, 3×3 and 4×4. The area and power stats represent the entire NoC and the average buffer costs. Table II, Table III and Table IV present the results of the synthesis analysis.

Note the *Optimized Buffer* solution presents a lower area and power impact than the *Extended Buffer* scheme in all cases. The increase of the *Optimized Buffer* costs in NoCs with sizes 3×3 and 4×4 are underneath 16%, unlike the *Extended Buffer* version that exceeds 30%. Therefore, the *Optimized Buffer* version proves to be an economical choice to protect buffers using ECCs. This version keeps the same memory size as *Regular Buffer* but includes a logic to manage the two stored data types: packet's flits and their redundancies.

TABLE II. Synthesis Analysis for a 2×2 NoC Phoenix.

Design	Level	Area (μm^2)	Increase (%)	Power (mW)	Increase (%)
Regular buffer	NoC	123122	0	2.97	0
	Buffer	5537	0	0.13	0
Extended buffer	NoC	164260	33.4	4.08	37.4
	Buffer	7504	35.5	0.19	42.5
Optimized buffer	NoC	139551	13.3	3.75	26.2
	Buffer	6322	14.2	0.15	13.4

TABLE III. Synthesis Analysis for a 3×3 NoC Phoenix.

Design	Level	Area (μm^2)	Increase (%)	Power (mW)	Increase (%)
Regular buffer	NoC	280352	0	6.53	0
	Buffer	5471	0	0.13	0
Extended buffer	NoC	372838	32.9	8.72	33.5
	Buffer	7441	36.0	0.18	35.8
Optimized buffer	NoC	317289	13.2	7.61	16.5
	Buffer	6341	15.9	0.15	16.8

TABLE IV. Synthesis Analysis for a 4×4 NoC Phoenix.

Design	Level	Area (μm^2)	Increase (%)	Power (mW)	Increase (%)
Regular buffer	NoC	501468	0	11.50	0
	Buffer	5506	0	0.13	0
Extended buffer	NoC	665675	32.7	15.00	30.4
	Buffer	7472	35.7	0.17	32.8
Optimized buffer	NoC	544350	8.55	12.90	12.1
	Buffer	6090	10.6	0.15	17.2

B. Error Coverage Analysis

This analysis inserts MCUs in the *Optimized* and *Extended* buffer versions, applying patterns of adjacent errors from 1 to 4 error scenarios, according to [13], which represents the most of error cases in memories. For each scenario, we generated 10000 error patterns randomly. This amount of error situations is enough to have an accurate error coverage analysis of each scenario. The testbench was developed in VHDL and focused on a single buffer connected to a fault injector block and a correction monitor, excluding the rest of the system. The simulation was done in Mentor Graphics ModelSim.

After the buffer is fulfilled, the fault injector applies the selected error pattern in a random buffer position. Then, the buffer is emptied, and the correction monitor analyzes the error correction capability. Table V presents the results for Error Coverage Analysis of both versions and illustrates the efficiency emerged in the fault tolerance experiments, considering the number of bit-flips corrected per event using ExHamming.

For the 2 errors pattern, most of the errors occurred in different stored flits, providing a high correction capability. As the error pattern becomes more aggressive (i.e., 3 and 4 errors), the occurrence of double errors in the same word increases, decreasing the correction rate. Notice the *Optimized Buffer* is also capable to correct a few 4 errors patterns, differently from the *Extended Buffer*. This result can be justified due to the better arrangement of the redundancy bits, provided by the *Optimized Buffer*, allowing the occurrence of single errors in different redundancies.

TABLE V. CORRECTION RATE OF EACH FAULT SCENARIO.

Scenario	Correction Rate (%)	
(number of faults)	*Optimized Buffer*	*Extended Buffer*
1	100.00	100.00
2	82.03	79.05
3	44.55	38.57
4	3.36	0.00

C. *Performance Evaluation*

We described in VHDL-RTL the injectors and receptors of traffic and connected them to each routers' local port. Using Mentor Graphics ModelSim, we simulated the injection of 50000 packets per core with rates from 15% to 44%. Fig. 5 shows the latency results regarding the injection rate, including the variation percentage between both solutions. Fig. 6 shows the accepted traffic following the same parameters as Fig. 5.

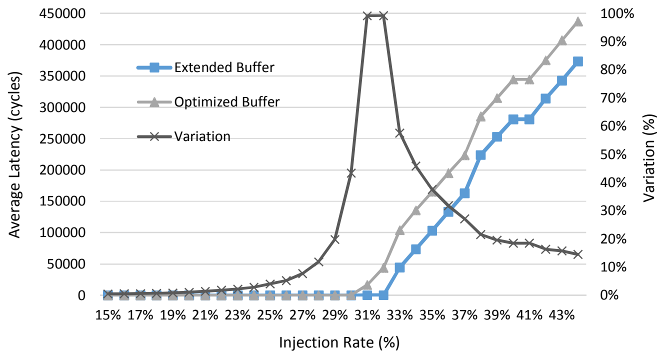

Fig. 5. Chaos Normal Form (CNF) of the average latency.

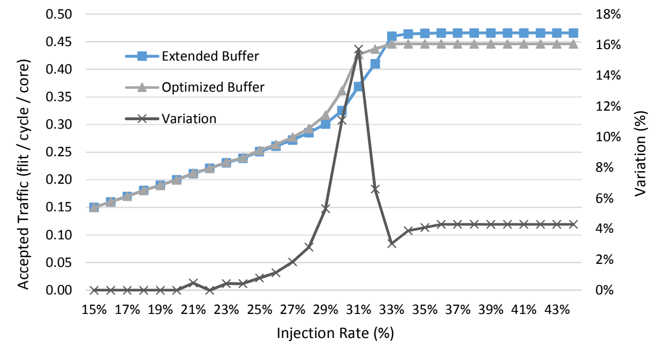

Fig. 6. CNF of the accepted traffic.

The variation curve of Fig. 5 shows the average latencies between the *Optimized Buffer* and *Extended Buffer* versions are closer from 15% to 23% of the injection rate. Starting from 23% to 31% of injection rate, which is the traffic saturation point for the *Extended Buffer* version, the proportional latency of the *Optimized Buffer* version increases. Finally, from the saturation point, both latencies increase meaningfully, but the relative difference of the latency is reduced. Fig. 6 shows that this behavior is a consequence of the acceptable traffic of each version. As the *Optimized Buffer* has a reduced storage capacity, it is expected that its accepted traffic is lower than the *Extended Buffer* version. Therefore, resulting in an earlier traffic saturation than another version. The spikes presented in Fig. 5 and Fig. 6 are the saturation moments in a range of injection rate (from 29% to 31%). Even with this variation, the *Optimized Buffer* version shows a small performance decrease.

IV. CONCLUSION

This paper proposes a fault-tolerant buffer architecture for NoCs applying ExHamming in order to enhance the trade-off between reliability and buffer's area and power resources. This buffer is called *Optimized Buffer*, which covers all the data stored. Synthesis Cost Impact analysis shown that the *Optimized Buffer* proposed presents better cost results than the *Extended Buffer* scheme.

The experiments of Error Coverage Analysis reveal that the proposed structure was also able to marginally outperform the *Extended Buffer* scheme in all MCU error patterns. Finally, the results reveal a short performance loss between the two versions, being slightly noticeable after the network saturation.

REFERENCES

[1] L. Benini, G. De Micheli. *Networks on chips: A new SoC paradigm*. **Computer**, v. 35, n. 1, pp. 70-78, Jan. 2002.

[2] Salaheldin, K. Abdallah, N. Gamal, H. Mostafa. *Review of NoC-based FPGAs Architectures*. **IEEE International Conference on Energy Aware Computing Systems & Applications (ICEAC)**, pp. 1-4, 2015.

[3] S. Jiang et al. *Study of Fault-Tolerant Routing Algorithm of NoC based on 2D-Mesh Topology*. **IEEE International Conference on Applied Superconductivity and Electromagnetic Devices (ASEMD)**, pp. 189-193, 2013.

[4] S. Khan et al. *Comparative analysis of network-on-chip simulation tools*. **IET Computers & Digital Techniques**, v. 12, n. 1, pp. 30-38, Sep. 2017.

[5] W. Wolf et al. *Multiprocessor System-on-Chip (MPSoC) Technology*. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 10, pp. 1701-1713, Oct. 2008.

[6] P. Ferreyra et al. *Failure map functions and accelerated mean time to failure tests: New approaches for improving the reliability estimation in systems exposed to single event upsets*. **IEEE Transactions on Nuclear Science**, v. 52, n. 1, pp. 494-500, Feb. 2005.

[7] A. Achballah, S. Othman, S. Saoud. *Problems and challenges of emerging technology networks-on-chip: A review*. **Microprocessors and Microsystems**. v. 53, pp. 1-20, Aug. 2017.

[8] C. Chen, M. Hsiao. *Error-correcting codes for semiconductor memory applications: A state-of-the-art review*. **IBM Journal of Research and Development**, v. 28, n. 2, pp. 124-134, Mar. 1984.

[9] M. Radetzki, C. Feng, X. Zhao, A. Jantsch. *Methods for fault tolerance in networks-on-chip*. **ACM Computing Surveys (CSUR)**, v. 46, n. 1, article 8, Oct. 2013.

[10] J. Silveira et al. *Scenarios Preprocessing Approach for Reconfiguration of Fault-Tolerant NoC based MPSoCs*. **Microprocessors and Microsystems**, v. 40, pp. 137-153, Feb. 2016.

[11] F. Silva et al. *Evaluation of multiple bit upset tolerant codes for NoCs buffering,* **IEEE Latin American Symposium on Circuits & Systems (LASCAS)**, pp. 1-4, 2017.

[12] L. Leem et al. *ERSA: Error Resilient System Architecture for Probabilistic Applications*. **Design, Automation & Test in Europe Conference & Exhibition (DATE)**, pp. 1560-1565, 2010.

[13] C. Ogden, M. Mascagni. *The Impact of Soft Error Event Topography on the Reliability of Computer Memories*. **IEEE Transactions on Reliability**, v. 66, n. 4, pp. 966-979, Dec. 2017.