PARITY CHECK FOR M-OF-N DELAY INSENSITIVE CODES

Julian Pontes[1,2], Ney Calazans[1], Pascal Vivet[2]

Faculty of Informatics - FACIN, - PUCRS[1]
Porto Alegre, RS, Brazil
ney.calazans@pucrs.br

CEA-LETI[2]
Grenoble, France
{julian.hilgembergpontes, pascal.vivet}@cea.f

***Abstract*—The advance in deep submicron technologies brings new constraints to circuit design such as variability and sensitivity to soft errors. Asynchronous networks on chip can help coping with some of these constraints due to the timing robustness of design paradigms such as the quasi delay insensitive one. A relevant problem of current fully asynchronous networks on chip is the lack of mechanisms to provide error detection and correction in asynchronous data communication. This work proposes a parity scheme applicable to m-of-n delay insensitive codes, which is capable to correct errors caused by single event effects in delay insensitive communication architectures. The proposed mechanism was evaluated in a 65nm technology where it is able to correct 98% of the errors caused by single event effects with a low overhead in terms of area, power and performance.**

***Index Terms*—soft error, single event effects, GALS, delay insensitive codes, network on chip, error correction.**

I. INTRODUCTION AND RELATED WORK

Synchronous circuits work at a frequency defined according to worst case timing assumptions, plus some extra timing margin to compensate effects such as clock skew and jitter, environmental effects and variability. Due to such aspects, the aggressive scaling of a unique clock to increasingly higher frequencies is no longer feasible in advanced deep submicron technologies [10].

The clock distribution restriction has led to the popularization of the Globally Asynchronous Locally Synchronous (GALS) paradigm, which allows the construction of complex systems on chip (SoCs) with local synchronous islands interconnected by asynchronous communication mechanisms [6]. Networks on chip (NoCs) are a trend in the design of such components, since they facilitate the scalability and exploration of parallelism, when compared to bus based communication architectures. Additionally, asynchronous NoCs have recently shown their benefits when compared to their synchronous counterparts to build future many-core architectures, in terms of both performance and power [7].

One of the next challenges for such asynchronous communication architectures is reliability, in the form of robustness to single event effects (SEEs). SEEs occur when particles hit the silicon, generating electrons-hole pairs [11]. If the victim node is in reversed bias, the electrical field of this node can collect the generated charge and create a bit flip. Technology downscaling continuously increases the logic sensitivity of silicon devices to such effects [11]. Asynchronous circuits have shown better response under SEEs than their synchronous counterparts [4] [16].Contrary to what happens in synchronous circuits, delay variations induced by radiation usually have no impact on asynchronous quasi-delay insensitive (QDI) circuits. However, bit flips may corrupt data transmissions and stall the circuit with no recovery solution.

In synchronous NoC implementations, data protection can employ retransmission schemes applicable between routers and end-to-end (between senders and receivers) [8] [12]. Another way to prevent data errors in NoC links is using Forward Error Correction (FEC) as proposed e.g. in [17] and [8]. Yu and Ampadu [20] present router-to-router and end-to-end data correction techniques. However, due to the migration from fully synchronous to GALS circuits, asynchronous NoCs are becoming a usual communication architecture choice. Asynchronous implementations are able to deliver high throughput with less dynamic power consumption. Moreover, immunity to delay variations in NoCs that employ DI Codes make then a good option for advanced node technologies. However, the asynchronous NoC implementation can be very sensitive to bit flips caused by soft errors and there are just a few works that treat error correction in DI encoding.

The work of Agyekum and Nowick [1] proposes an unordered DI code enabling two-bit error detection and one-bit error correction capabilities. However, this code is difficult to implement in a fully QDI way, which makes it hard to use in fully asynchronous NoCs. Bainbridge and Salisbury [3] present a set of techniques to apply to QDI Networks on chip links, particularly for links based on m-of-n encoding, to reduce glitch sensitivity. But these techniques are limited to filtering some glitches at the NoC link wires. None of them is adequate for soft errors and none offers data error correction. Pontes et al. [15] proposed the Temporal Redundancy Delay Insensitive Code (TRDIC) Code. This code is obtained by converting a 1-of-n code to a 2-of-n+1 code, by using the extra valid codewords of the new code to add temporal redundancy. TRDIC has the ability to filter, detect and correct errors caused by glitches in asynchronous data transmission. However, TRDIC requires around 180% of area overhead, due to the complexity of completion detectors for 2-of-n codes.

The most common DI code used in asynchronous NoCs is 1-of-n [2] [5]. A problem with NoCs that use this kind of code

978-1-4799-0662-8

is the absence of a data error correction scheme applicable to them. Since each m-of-n codeword usually carries more than one bit information, an error in the m-of-n codeword can affect several bits. Thus, traditional correction techniques can be less efficient in asynchronous NoCs. This work proposes a new error correction/detection scheme based on parity computation. It is applicable to asynchronous NoCs in GALS SoCs as that described in [7]. The proposed scheme can be applied to any m-of-n code with low area overhead. Here, the NoC-based communication in presence of single event effects is evaluated, but the proposed error correction scheme can also be used for intra/extra chip point to point, bus-based communication to solve soft errors from radiation and crosstalk.

The rest of this document is organized in four sections. Section II approaches the problem of soft errors in delay insensitive (DI) data communication, with focus on m-of-n codes. Section III proposes the parity scheme for m-of-n DI encoding. Section IV presents and discusses the conducted experiments and obtained results of the NoC soft error evaluation. Finally, Section V presents some conclusions and directions for future work.

II. Soft Errors in m-of-n DI Codes

A. Introduction to QDI Data Transmission

The most common way to encode data in digital domains is using regular binary codes where each logical bit value is associated to a voltage level. However, more sophisticated codes can be used to meet data transmission constraints such as clock recovering, energy consumption and error correction and detection.

In asynchronous circuits, the target characteristic of codes is often to guarantee delay insensitivity during data transmission. Data transmission can be achieved in a delay insensitive way when the code is *unordered*. A code is said unordered when a codeword that is part of this code is not covered by any other codeword in the same code [19]. According to [1], a codeword $X=x_1x_2...x_n$ covers another codeword $Y=y_1y_2...y_n$ if and only if, for each bit position i, if $y_i = 1$ then $x_i = 1$.

M-of-n codes are a subclass of *unordered* codes. In these, each codeword takes exactly n wires. A valid m-of-n codeword has m of these wires equal to *1*. Table 1 shows an example, the 1-of-4 code, represented by all of its valid codewords (except for the first line of the Table, which contains an invalid codeword). In this code, four wires are used to encode two bits of information. In each valid codeword exactly one of the four wires is 1. M-of-n codes, as all unordered codes, have the inherent ability to detect the so-called unidirectional errors [19]. This property can be explored in communications systems where retransmission is an option. However, for communication systems where data correction is required, this class of codes requests for even more redundancy. As a result, the final code can present a very low code density. The proposed code can solve this problem at very low cost in area and code density, as demonstrated here.

Table 1 – The 1-of-4 code. The spacer is not a valid codeword.

Value	A3	A2	A1	A0
Spacer	0	0	0	0
“00”	0	0	0	1
“01”	0	0	1	0
“10”	0	1	0	0
“11”	1	0	0	0

A DI communication is coordinated by a handshake protocol [18]. Here, only the 4-phase handshake protocol is considered, but the proposed error correction scheme is extendable to 2-phase DI data transmission as well. To perform a DI data transfer using a 4-phase protocol, it is necessary the definition of a special codeword to represent the absence of data, which is called *Spacer* in Table 1. A spacer here is encoded with all wires at logic ‘0’. Figure 1 shows examples of 1-of-4 data transfers using the four-phase handshake protocol. The protocol starts with a valid data (“0001”), which means that the data signals carry a valid codeword. When the receiver samples the data it asserts an *acknowledgement signal* (Ack). The sender then produces a spacer, indicating the end of transmission and the Receiver answers with another Ack assertion. A spacer occurs between every two valid codewords, to complete the 4-phase protocol, guaranteeing the DI property.

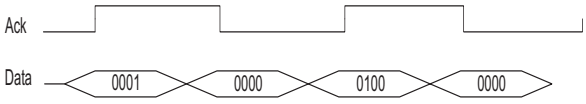

Figure 1- Four-phase handshake protocol.

Asynchronous QDI data links are built through pipelines using two main components: asynchronous registers and completion detectors (CD). Asynchronous registers are implemented using C-element as memory elements. A C-element outputs 1 when all its inputs are 1 and outputs 0 when all its inputs are 0. Otherwise, it keeps its prior output value. Other relevant circuits in QDI circuits are completion detectors. It is responsible to detect if there is valid data available at the output of a register (or any block). Figure 2 shows the implementation of an asynchronous 3-stage pipeline for the 1-of-4 code and four-phase protocol using weak conditioned half buffers. For the 1-of-4 pipeline, the completion detector comprises a single four-input NOR gate.

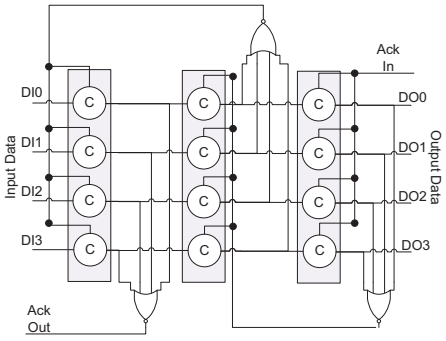

Figure 2 – 1-of-4 QDI pipeline using weak conditioned half buffer.

B. SEE impact on C-elements

To precisely analyze the behavior of QDI pipelines under SEEs, it is first required to analyze the behavior of the C-elements under radiation. Figure 3 summarizes this behavior with a state graph that describes the behavior of a simple 2-input C-element under the presence of radiation. The C-element can present single event transients (SETs) or single event upsets (SEU). The effect depends on the state of the victim C-element when it is hit by a particle. In states 000 and 111, the C-element has a continuous electrical path from inputs to the output. In this case, a particle hit can generate a SET (a transient fault). In states 010, 100, 011 and 101, the C-element acts as a memory element. In these cases, radiation can cause an SEU (an upset or bit flip fault).

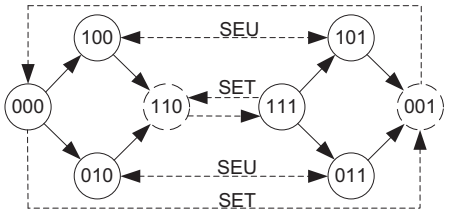

Figure 3 - State transition graph of a C-element in presence of SEEs.

C. SEE impact on DI data transmission

To better understand the behavior of a soft error in QDI data transmission, the four-phase protocol can be sliced in time windows as proposed in [15]. Figure 4 shows the timing and SEE analysis on a 4-phase protocol for an m-of-n weak conditioned half buffer. For each phase of the protocol, the timing description appears on top side of the figure. The *Best Case Data Delay* indicates the fastest path between the previous stage and the register input. The *Data Skew* is the timing difference between the fastest and the slowest bit in the datapath. The Acknowledge Delay (*Ack Delay*) is the time needed from data propagation to completion detection, plus the detection delay. The observed SEEs in each window are the result of the C-element behavior.

In the *Best Case Delay* timing window, the only possible SEE is the occurrence of an SEU↑, due to the behavior of an SEE in a C-element, when C-element inputs are different. This event will generate an *Incomplete Data* (ID). An incomplete m-of-n data has *k* wires equal to 1, where k < m.

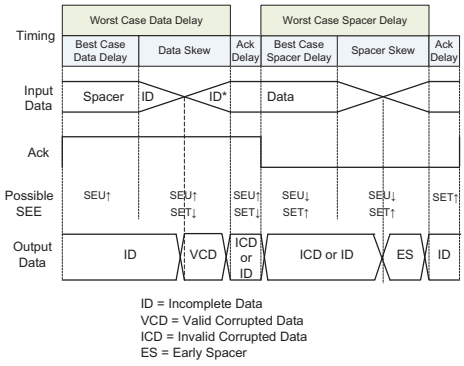

Figure 4 - Timing and SEE details of m-of-n four-phase protocol.

The *Data Skew* timing window can be divided in two windows. These are delimited by the Input Data State. If *m-1* bits of the Input Data already switched (ID*), then the encoding reaches an *excited state*, otherwise the inertial property of the code will keep filtering any event at the Input Data. The *Data Skew ID** timing window is the only window where a Valid Corrupted Data (VCD) can happen. A VCD is an erroneous data transmission where transmitted data consists in a valid m-of-n code, i.e. with m wires equal to 1. Since the data is already switching during this window, it is expected that the remaining data wire switches before the *Ack In* signal arrives (event that closes the register for data propagation). If the data wire arrives before the acknowledge signal, the result will be an Incomplete Corrupted Data (ICD) instead of a VCD. An ICD presents the correct data information plus an extra bit that makes the DI code invalid (m+1-of-n). The invalid code presented by a corrupted data is able to cross pipeline stages and arrive to the final data receiver, since it is detected by a regular completion detector implementation.

The VCD can be removed at design time by guaranteeing that the data skew is smaller than the *Ack In* propagation. Data skews are usually rather small, since all data signals are generated in the rising edge of the Acknowledge signal. The environment must always detect and sample data before acknowledging it. Looking inside the pipeline, it is possible to note that the Acknowledge propagation will be the time to cross the next stage register, plus the completion detection time. This gives enough time to generate the missing data wire.

In 1-of-n DI codes, the Hamming distance between the spacer and all the data symbols is always one. In this way, data links implemented with 1-of-n codes are always in the *excited state*. This means that when transmitting a spacer, an SEE↑ (i.e. a soft error that produces a logic transition from '0' to '1') is able to generate new data. When transmitting some valid data, an SEE↓ (a '1' to '0' transition) is able to clear the data and generate a spacer. In addition, the Incomplete Data presented at Figure 4 is replaced in 1-of-n code by an Early Spacer or an Early Data. These early indications alter the four-phase protocol sequence and can cause stall in the protocol. The four-phase protocol hardening is not treated in this work. Here, just the error correction for the data content is considered. Other types of errors require different hardening techniques.

III. GALS Parity Check in m-of-n Codes

The error detection of α or fewer bits in a data transmission is only possible when the Hamming distance between codewords is not smaller than $\alpha+1$. The distance between any two valid m-of-n codewords is equal to or bigger than 2. In this way, m-of-n codes are intrinsically able to detect single bit errors (ICD) in data transmission. To overcome ICD data errors, this property can trigger packet retransmission requests, but these may result in unacceptable communication latencies, due to natural packet retransmission delays or increased NoC congestion effects. Analyzing the timing window of the m-of-n four-phase data transmission in Figure 4, it is possible to observe that the ICD is the most common data integrity problem. Thus, an ICD data error correction scheme can

mitigate such problems. This work proposes the use of a parity scheme for m-of-n codes combined with the intrinsic property of unordered codes to detect single data errors for correcting all ICDs and detect all VCDs in asynchronous QDI NoC circuits.

A. System Architecture

Figure 5 shows details of a communication architecture for a GALS SoC based on an asynchronous NoC that employs a parity scheme. The sender IP is synchronized with a parity encoder while the receiver IP is synchronized with a parity decoder. There is no relation between sender and receiver clocks and the synchronization mechanism required in this GALS SoC is omitted. The architecture is quite similar to the MAGALI GALS chip implementation described in [7].

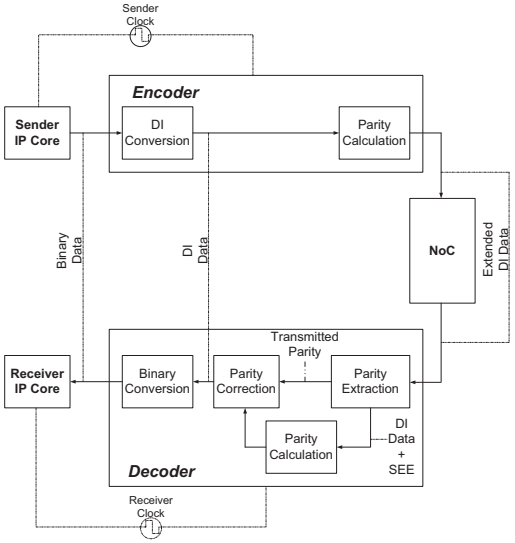

Figure 5 - Parity communication architecture.

The data (Binary Data) exchanged between sender and receiver IP Cores is structured in packets transmitted through the asynchronous NoC [13]. This NoC is available in two distinct DI code-based implementations, dual-rail and 1-of-4, both employing four-phase handshake protocols. Internally, the NoC uses XY routing for packet transmission in wormhole switching mode. Each packet comprises two fields: a header with routing information and a payload.

The parity *Encoder* receives binary data and converts each datum to an m-of-n codeword in the *DI Conversion* module. The main novelty of the proposed scheme is that *Parity Calculation* is performed on generated m-of-n codewords, rather than on binary data. Since each m-of-n codeword can transport more than one bit, a single bit flip in the m-of-n codeword during transmission can generate multi-bit errors in the binary data. In this way, a mechanism for error correction of m-of-n codewords can be more effective than an equivalent process for binary codewords. In this work, each binary data originated from the Sender represents an elementary portion of a packet (in fact, a flit) to be transmitted through the asynchronous NoC.

The parity *Decoder* is divided into four modules. *Parity Extraction* removes the transmitted parity bits from each flit and forwards these to the *Parity Correction* unit. The *Parity Calculation* module receives each flit and applies the same parity generation method used in the *Encoder*. The *Parity Correction* module receives the flit and the two parity vectors. Based on this information, it can detect/correct errors in m-of-n codewords. After correction, *Binary Conversion* occurs, and data are delivered to the receiver.

B. The m-of-n Parity Encoding

Parity check is a block code where redundant bits are added to a message to enable error detection and/or correction. The use of unordered codes also enables error detection in data transmission. Here, parity check is used associated to unordered codewords to detect and correct data errors in DI data transmission. It is important to note that since an error in a m-of-n codeword can affect several data bits, the use of parity alone is not able to allow correcting data errors. For example, if the codeword 0001 is transmitted but the codeword 1001 is received, it is not possible to determine if the correct data is "00" or "11", since both have the same parity.

This work assumes that the communication system is a GALS SoC where all IP Cores are synchronous, while the communication architecture is formed by asynchronous channels using some m-of-n code. In the synchronous side, binary code is used. This means that each datum is represented as a binary data vector $d = [a_0, a_2, ..., a_{k-1}]$, where for all a_i, $a_i \in \{0, 1\}$. To perform data communication through the NoC, data is translated to an m-of-n codeword using a bijective function denoted DI(d). The binary data vector has length L(d) equal to k. The length is defined as the number of bits in the codeword. Of course, in this case the number of possible different codewords that can represent some data is 2^k. The maximum number of distinct codewords that can be encoded in a single m-of-n vector is defined by the combination:

$$C_{n,m} = \frac{n!}{m!(n-m)!}. \quad (1)$$

Therefore, the minimum number of m-of-n codewords S(d) that is needed to cover all binary data vectors in d is:

$$S(d) = ceiling\left(\log_{C_{n,m}} 2^k\right) \quad (2)$$

Any data with L(d)=k is transformed into an m-of-n data matrix (flit) $F = [f_{i,j}], i=1,...,s, j=1, ...,n$, where s = S(d) and **n** is the number of wires of the m-of-n code.

For example, the binary vector $d = [0111100110111100]$ with Length L(d) = 16, when converted to the 1-of-4 encoding by DI(d) generates a 1-of-4 data flit matrix F, shown in (3) below. Here, the number of columns is equal to the number of wires in the m-of-n code (four in this example, which uses the 1-of-4 code) while the number of lines is S(d)=8, and for all f_{ij}, $f_{ij} \in \{0,1\}$. Starting with (3), a parity vector P = (p_0, p_2, …, p_{n-1}) with length L(P) = n is generated with a XOR operation of all elements in a same column as follows:

$$\forall_i, P_i = f_{i,0} \oplus f_{i,1} \oplus ... \oplus f_{i,s-1}.$$

Applying the XOR operation to matrix (3), the parity vector P is computed as: P(F) = (1001). After generation, the parity

vector is translated to equivalent m-of-n codewords. The translation uses the bijective function DI(P). The result is S(P), a new set of m-of-n codewords, concatenated to Matrix F to generate the extended flit matrix EF. In matrix (4), the last two rows are the equivalent 1-of-4 codes DI(P) of the parity vector.

$$F = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (3) \qquad EF = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (4)$$

It is important to note that the parity scheme adopted here associates one single parity bit to each column of the matrix. More elaborate parity implementations can be applied where more than one bit is used for each column. This potentially increases the correction capability of the parity scheme. The proposed code is one special case of the product codes where the unordered codes identifies the row containing the error in the matrix while the parity isolates the column. In this way, the element in the matrix that contains the error can be defined and corrected.

C. The m-of-n Parity Decoding

Parity decoding is achieved based on the transmitted parity and the recalculated parity conducted at the receiver side. If the transmitted parity contains an ICD, correction is not performed. Since only a single data error is considered in each flit, the error in Parity codeword means that the data is error-free. If an ICD is detected at the Parity codeword and another ICD is present in any data codeword at the same flit, then an error is indicated to the IP Core and a retransmission can be requested.

When the transmitted parity does not carry any error, an XOR operation is performed between the two parity vectors (received and recalculated). If the result is non-zero, there are errors in the flit. The position of the '1' in the resulting XOR indicates the position of the error, that is, it indicates which column of the F matrix (3) contains the error. To identify the m-of-n code that contains the error, the Rows of the F matrix are verified to identify an ICD. In this way, the fault in the unordered property of the code indicates the m-of-n codeword that contains the error (isolating the row) while the parity indicates which wire carries the error (isolating the column). When the victim codeword is detected, the operation defined in the truth table depicted in Table 3 is performed, wire by wire.

Table 3- Parity correction operations.

Data Wire	Parity	Corrected Data
0	0	0
0	1	0
1	0	1
1	***1***	***0***

Figure 6 shows an example of data correction in an 1-of-4 code with a single ICD error.

If a single flit is victim of two or more errors in data codewords in different columns, the parity scheme is still able to correct the error. In this way, in a m-of-n data transmission the parity can fix until *n* errors since these errors appear in different columns of the transmitted matrix. The VCD error alters the parity regeneration in two different columns for the 1-of-n code since the VCD consists in a two-bit error. In this case, the row error isolation is not possible. Thus, data correction is not possible, but error detection can be achieved, since the parity indicates an error.

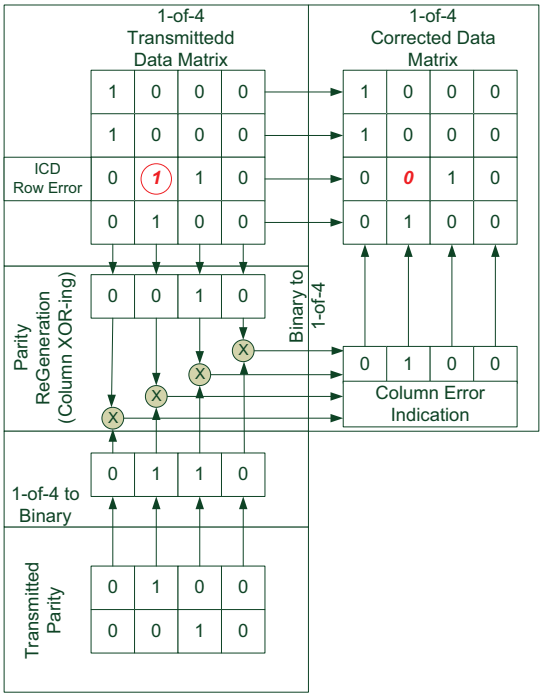

Figure 6 –Single ICD Data Error Correction example.

IV. CONDUCTED EXPERIMENTS AND RESULTS

Soft error evaluations were performed in a fully asynchronous 4x4 mesh NoC [14] using the characterization method proposed in [13]. This NoC uses the 1-of-4 code and four-phase handshake protocol. It was designed to transport 32-bit flits. Thus, sixteen 1-of-4 codewords are needed to carry a flit. One additional codeword is used to control the packet transfer protocol. Thus, seventeen 1-of-4 codewords are needed to transmit a flit. The NoC version with parity corrections uses four extra bits for parity. Thus, two extra 1-of-4 codewords are necessary and nineteen 1-of-4 codewords are needed for a flit.

The evaluation traffic scenario is based on the MultiMedia System (MMS) application described in [9]. The presented results were obtained from a commercial 65nm technology working at 1V, 25ºC and typical transistor models.

Figure 7 shows the error classification as a function of the resulting DI code corruption, when the NoC is under a fault injection test case with charge injection of 70fC.

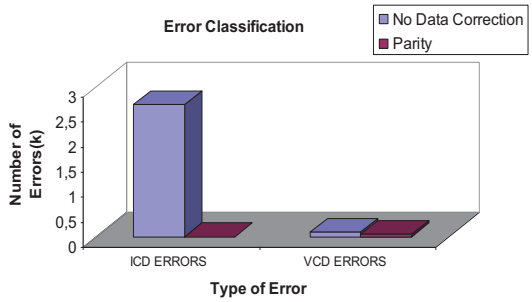

Figure 7 - DI error classification.

Results show that the ICD errors are the most common, as expected. Since the proposed parity scheme is capable to correct all the ICDs, in the parity evaluation no ICD was detected at the receiver side. The number of VCDs was kept the same. It is important to note that the parity scheme can detect VCD errors. Thus, residual VCDs can be solved by a retransmission scheme, for example.

Table 4 shows the area and static power comparison of the asynchronous NoC and the NoC with the additional parity codewords. The area overhead of 12% is far below the 180% presented by the TRDIC scheme proposed in [15] and is capable to correct the same class of errors. The area results of sixteen encoders and sixteen decoders are also shown. These sixteen Encoders/Decoders are necessary in the 4x4 NoC example used in the evaluation.

Table 4 - Area overhead in the asynchronous NoC.

Implementation	Area (sq. mm)	Area Overhead
NoC	1.31000	-
NoC + Parity	1.47000	12.9%
Encoder	0.17936	1.37%
Decoder	0.68784	5.2%

Summing up the NoC, encoder and decoder areas, the total system overhead for the 4x4 test case is 19.47%. This area overhead is ten times smaller than that of the TRDIC technique proposed in [15], for example. It is important to note that the NoC area occupied by the parity bit is independent on the number of bits in the flit. However, the area increases with the number of wires of the m-of-n code. The latency overhead is around two clock cycles, one for parity calculation and another for decoding and correction. The throughput of the NoC it is not affected by the extra parity codewords, since they are sent in parallel with the data flit.

V. Conclusion and Future Works

In this work a new scheme for error detection and correction for m-of-n DI Codes was proposed and evaluated. The correction scheme is based on parity and can detect all 2-bit flips, referred here as VCD errors. It can also correct all single bit flips, referred here as ICDs. The proposed scheme can be applied to any GALS system with asynchronous communication employing m-of-n codes. However, due to its simplicity and low area overhead, it is well adapted for end-to-end data correction in GALS SoCs based on asynchronous NoCs. Future works include the use of the proposed parity scheme with different m-of-n codes. The hardening of the control circuitry of the NoC against soft errors is also an ongoing work.

References

[1] Agyekum, M. Y.; Nowick, S. M. “An error-correcting unordered code and hardware support for robust asynchronous global communication.” In: DATE’11, pp. 765-770, 2011.

[2] Bainbridge, J.; Furber, S. “Chain: A Delay-Insensitive Chip Area Interconnect.” IEEE Micro, 22(5), pp. 16-23, Sep.-Oct., 2002.

[3] Bainbridge, W. J.; Salisbury, S. J. “Glitch sensitivity and defense of quasi delay insensitive network-on-chip links.” In: ASYNC’09, pp. 35-44, 2009.

[4] Bastos, R. P.; Sicard, G.; Kastensmidt, F.; Renaudin, M.; Reis, R. “Asynchronous circuits as alternative for mitigation of long-duration transient faults in deep-submicron technologies.” Microelectronics Reliability, 50, pp. 1241-1246, Nov. 2010.

[5] Beigné, E.; Clermidy, F.; Vivet, P.; Clouard, A.; Renaudin, M. “An Asynchronous NoC Architecture Providing Low Latency Service and its Multi-level Design Framework.” In: ASYNC’05, pp. 54-63, 2005.

[6] Chapiro, D. “Globally-Asynchronous Locally Synchronous Systems.” PhD Thesis, Stanford University, Oct. 1984, 134 p.

[7] Clermidy, F.; Cassiau, N.; Coste, N.; Dutoit, D.; Fantini, M.; Ktenas, D.; Lemaire, R.; Stefanizzi, L. “Reconfiguration of a 3GPP-LTE telecommunication application on a 22-core NoC-based system-on-chip.” In: NoCS’11, pp. 261-262, 2011.

[8] Frantz, A. P.; Cassel, M.; Kastensmidt, F. L.; Cota, E.; Carro, L. “Crosstalk and SEU-Aware Networks on Chips.” IEEE Design & Test of Computers, 24(4), pp. 340-350, Jul.-Aug. 2007.

[9] Hu, J.; Marculescu, R. “Energy- and performance-aware mapping for regular NoC architectures.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(4), pp. 551-562, Apr. 2005.

[10] International Technology Roadmap for Semiconductors, http://www.itrs.net/, 2011.

[11] Mukherjee, S. “Architecture Design for Software Errors.” Morgan Kaufmann Publishers, Burlington, 2008. 337p.

[12] Murali, S.; Theocharides, T.; Vijaykrishnan, N.; Irwin, M.J.; Benini, L.; De Micheli, G. “Analysis of error recovery schemes for networks on chips.” IEEE Design & Test of Computers, 22(5), pp. 434- 442, Sept.-Oct. 2005.

[13] Pontes, J.; Vivet, P.; Calazans, N. “An Accurate Single Event Upset Digital Design Flow for Reliable System Level Design”. In: DATE’12, pp. 224-229, 2012.

[14] Pontes, J. J. H.; Moreira, M. T.; Moraes, F. G.; Calazans, N. L. V.: “Hermes-AA: A 65nm Asynchronous NoC Router with Adaptive Routing”. In: SOCC’10, pp. 493-498, 2010.

[15] Pontes, J.; Calazans, N.; Vivet, P. “Adding Temporal Redundancy to Delay Insensitive Codes to Mitigate Single Event Effects.” In: ASYNC’12, pp. 142-149, 2012.

[16] Rahbaran, B.; Steininger, A. “Is asynchronous logic more robust than synchronous logic?” IEEE Transactions on Dependable and Secure Computing, 6(4), pp. 282-294, Dec. 2009.

[17] Rossi, D.; Angelini, P.; Metra, C. “Configurable Error Control Scheme for NoC Signal Integrity.” In: IOLTS’07, pp. 43-48, July 2007.

[18] Sparsø, J.; Furber, S. “Principles of Asynchronous Circuit Design – A Systems Perspective.” Kluwer Academic Publishers, Boston, 2001. 354p.

[19] Bose, B. “On Unordered Codes.” IEEE Transaction on Computers, 40(2), pp. 125-131, Feb. 1991.

[20] Yu, Q.; Ampadu, P. “Dual-Layer Adaptive Error Control for Network-on-Chip Links.” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20(7), pp. 1304-1317, Jul. 2012.