A Fault Prediction Module for a Fault Tolerant NoC Operation

Jarbas Silveira[1], Mathieu Bodin[2], João Marcelo Ferreira[1],
Alan Cadore Pinheiro[1], Thais Webber[3], César Marcon[3]

[1]LESC-DETI / Federal University of Ceará - Fortaleza, Ceará, Brazil - 90455-970
[2] Polytechnique Nice Sophia Antipolis, Département Electronique – Biot, France - 06410
[3]PPGCC / PUCRS - Porto Alegre, Brazil – 90619-900
E-mail: jarbas@lesc.ufc.br

Abstract

Each new production technology of integrated circuit (IC) drives more transistors area reduction, implying smaller and denser circuits. This scenario allows integrating several Processing Elements (PEs) into the same IC with efficient communication architecture such as the scalable topologies of Network on Chip (NoC). However, these newer production technologies introduce more defects in various parts of the IC that have to be detected and well corrected to prevent malfunction of the IC. This work presents the Fault Prediction Module (FPM), which presents low area consumption and a power circuit based on thresholds enabling to detect link quality, i.e. operating properly, operating with fault tendency or with permanent fault. Additionally, we show how to tune the FPM threshold parameter aiming to use this circuit as a mechanism with comprehensive fault model. The set of experimental results shows the effectiveness of our proposal.

Keywords

Fault-tolerance, NoC, latency evaluation

1. Introduction

A Network-on-Chip (NoC) is a communication architecture that plays a key role in the implementation of a highly integrated System-on-Chip (SoC). Mesh is the most popular NoC topology offering simple and regular structure, and small wire length being suitable to the tile-based design. Routers and links interconnect each Processing Element (PE) to other PEs placed into a tile.

Recent deep submicron technologies increase defective components, mainly due to process variability [1], electromagnetic interferences, charge injection or radiation [2], or even cross talk effect [3]. Therefore, it is essential to provide support for dynamic faults using a circuit that detects and corrects faults, and additionally providing statistics to a high-level layer (e.g. an operating system).

This work presents the Fault Prediction Module (FPM), which is a circuit that distinguishes both transient and permanent faults, and determines if there is fault tendency on a link. The fault detection model is based on thresholds, which are triggered according to three types of transmissions: (i) a transmission without error, which produces an acknowledgement (**ACK**); (ii) a transmission with corrected errors, which also produces an acknowledgement but it is represented as **ACKC** (i.e. an ACK with error correction); and (iii) an unsuccessful transmission due to the quantity of errors that are detected but not corrected, which produces a negative-acknowledgment (**NACK**). The main challenge of FPM design is how the threshold parameters have to be set for the correct detection of fault tendency and permanent faults. FPM is part of the fault-tolerant mechanism of Phoenix NoC [4], and each input link of NoC's router contains a FPM.

The paper is organized as follows. Section 2 provides an overview of Phoenix NoC's architecture providing the architectural context of FPM. Section 3 details FPM circuit and its overall behavior. In Section 4, we show and discuss experimental results. Finally, Section 5 concludes this work.

2. Phoenix NoC's Architecture

Phoenix is a distributed fault-tolerant architecture implemented over a NoC-based MPSoC platform. It comprises hardware part placed on each NoC router and software part (i.e. *OsPhoenix*) placed into the operating system of each PE [5], which is composed of a processor-memory pair [4]. In addition, Phoenix NoC employs a direct 2D mesh topology consisting of routers using bidirectional links to connect with other routers and PEs. The NoC employs tables for source routing decisions, and the *OsPhoenix* implements routing algorithms to fill the *Routing Table* according to the position of PEs and faulty links.

Figure 1 shows the Phoenix's router architecture, which includes mechanisms for packet routing and fault-tolerance.

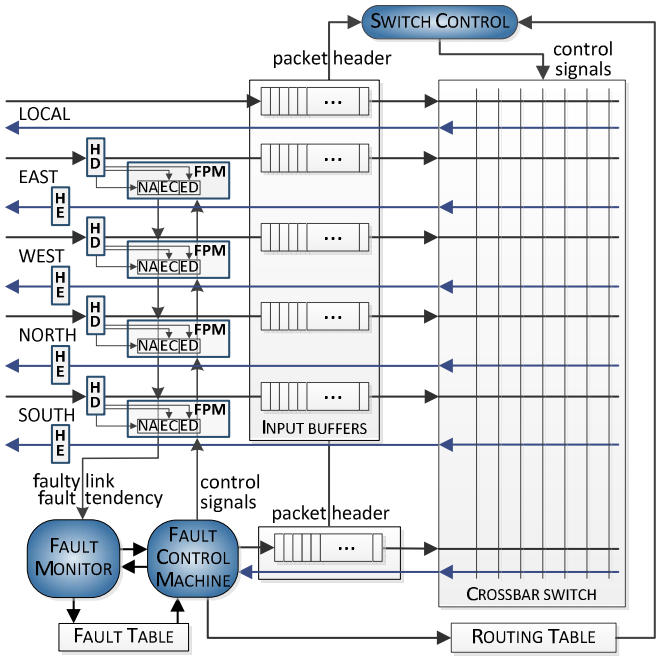

Figure 1: The basic components of Phoenix's router architecture.

The basic packet routing mechanism of Phoenix's router encompasses four set of components described next: (i) four bidirectional ports (NORTH, SOUTH, EAST and WEST) dedicated to interconnect routers, and a bidirectional port (LOCAL) that enables the communication between the router and its local PE. All input links contain configurable buffers used when packets congest the routing path; (ii) a *Crossbar Switch* that establishes unblocking connections between input and output ports; (iii) a *Routing Table* that implements the routing algorithm; and (iv) a *Switch Control* circuit that performs packets routing and arbitration according to the packet header and to the *Routing Table* content.

The fault-tolerance mechanism implemented in each router includes three types of circuits: (i) a fault detection and correction module containing a *Hamming Encoder* (HE), a *Hamming Decoder* (HD) and the FPM that are placed in each one of the bidirectional links; (ii) the *Fault Monitor* that

communicates with FPM to set links status on the *Fault Table* according to a two-level fault model; and (iii) the *Fault Control Machine*, which controls the *Fault Monitor* and the FPM, and communicates via control packets with the PE and with the *Fault Control Machine* of other routers.

Figure 2 depicts the two-level fault model implemented in Phoenix's architecture. Both levels store the operation status of the NORTH, SOUTH, EAST and WEST links. The first level contains two bits to inform whether the link is: (i) *not verified*, (ii) *operating properly*, (iii) *operating with fault tendency*, or (iv) *with permanent fault*. The second level complements the first one, providing extra information about the quality of link. This second level, which is physically placed inside each FPM, has counters with the operation status of the output link: (i) No Error (NE), (ii) Corrected Error (CE) and (iii) Detected Error (DE). In the example of Figure 2, the counters have different lengths in order to implement a window of events to capture probabilities of NE, CE and DE.

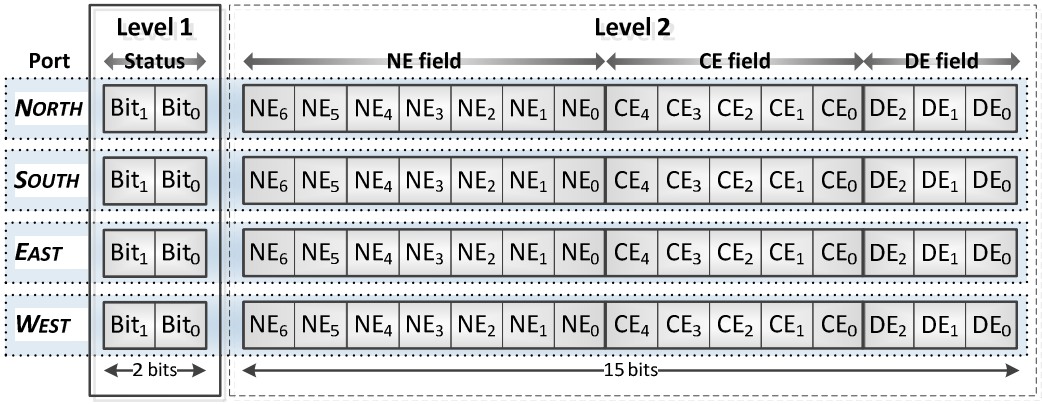

Figure 2: Two-level fault model of Phoenix: Level 1 - *Fault Table*; Level 2 - Link status counters (exemplified with 15-bit length).

Phoenix implements two complementary mechanisms for testing links quality: one static and other dynamic. Figure 3 depicts the MSC diagram of static link test, which starts with each *OsPhoenix* sending a test command (TEST_LINKS) through the router's local port to the *Fault Control Machine* that interprets this command and broadcasts a predefined test packet to all output ports.

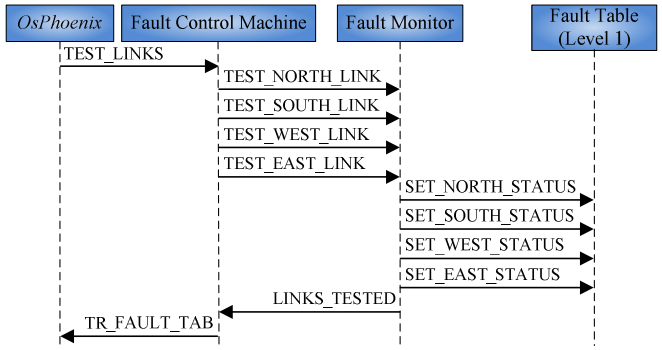

Figure 3: MSC diagram of the fault detection circuit.

When the input link of the neighbor router receives the test packet, it loops back a packet with the same information. Then, the *Fault Monitor* detects whether the link is faulty or not, sets this information on the *Faulty Table*, and informs this procedure to the *Fault Control Machine* that sends the *Faulty Table* to the *OsPhoenix* (TR_FAULT_TAB). The static mechanism sets only the Level 1, changing the status link from "*not verified"* to "*operating properly*" or "*with permanent fault*".

In order to accomplish the link test, each bidirectional link contains a HE and a HD to perform a dynamic strategy similar to the one used in [6], which identifies fault tendencies based on thresholds of ACKs and NACKs. The HD, placed on the input of the buffer, receives the data and the redundancy bits that were encoded by the HE in the adjacent router. The HD module can correct one bit flip and detect at most two faults in a data flit. Thus, the module informs the communication status by the counters NE, CE and DE, which performs the Level 2 of the fault model. The HE module generates the redundancy bits according to the flit that will be transmitted. FPM assumes a fault tendency or a permanent fault on link measuring the density of ACKCs or NACKs, respectively, in a given window of time. This fault tendency information is propagated to the *OsPhoenix* that may perform

high-level decisions to accomplish the fault-tolerance mechanism.

3. Fault Prediction Module (FPM) Description

FPM is a fault prediction circuit based on the architecture and functioning of a previous circuit described in [6]. FPM modifies the previous version inserting additional hardware to provide analysis of fault tendency and permanent fault. Figure 4 shows the basic architecture of FPM.

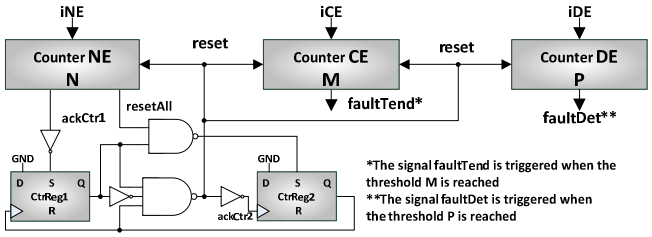

Figure 4: Architecture of FPM containing its main components.

Three counters and a reset circuit compose FPM. At each counter is associated a threshold. When the count reaches the threshold value, the counter fires a command signal (i.e. resetAll, faultTend or faultDet).

An event on signal iNE informs to the counter NE the occurrence of a successful flit reception (i.e. an ACK), implying an increment of the counter. When the counter NE reaches the threshold N, the counter produces a pulse on signal resetAll that initializes a reset sequence zeroing all counters (i.e. NE = CE = DE = 0). Details of reset sequence are described in [6]. The threshold N logically describes an observation window, where FPM may verify the quantities of ACKCs and NACKs, and consequently FPM may infer if the link is operating properly, if there are some transitory faults or the link operating in a situation considered as permanent fault.

An event on signal iCE informs the occurrence of a successful flit reception, but with the Hamming circuit correcting a single error (i.e. an ACKC), implying an increment of the counter CE. When the counter CE reaches the threshold M, the counter fires the command signal faultTend, notifying to the *Fault Control Machine* that the quantity of ACKCs received during the predefined windows indicates a fault tendency. Finally, an event on signal iDE informs the occurrence of an unsuccessful flit reception, i.e. the Hamming circuit detects, but does not correct, a double error characterizing occurrence of NACK. This event increments the counter DE and makes the router to require flit retransmission. Aiming to avoid retransmission delays, when the counter DE reaches the threshold P, it fires the command signal faultDet, notifying to the *Fault Control Machine* that the quantity of NACKs received during the observation windows indicates that it has to discard the link from the NoC's communication paths.

The *Fault Control Machine* informs to *OsPhoenix* any event of faultTend and faultDet. Additionally, these events may alter the Level 1 of the fault model according to the following decreasing priority of status: (i) *operating properly*, (ii) *operating with fault tendency*, (iii) *with permanent fault*. Therefore, a link marked as *operating properly* may be set to *operating with fault tendency* or *with permanent fault* in a presence of faultTend or faultDet events, respectively; but a link marked as *with permanent fault* only may change its status with a high-level command provided by *OsPhoenix*. Moreover, *OsPhoenix* sets all initial threshold values and may dynamically change them according to faults occurrence and high-level definitions, which are not discussed here.

It is important to notice that if the counters CE and DE do not reach their thresholds during the observation window the command signals faultTend and faultDet will not fire. Therefore, all occurrences of faults are perceived as transitory faults.

3.1 The Threshold Modeling

The use of thresholds aims to build an efficient circuit that captures permanent faults (i.e. fault tendency and permanent fault), discarding the transient ones. It has to consider architectural aspects such as the size of counter, and dynamic aspects as the definition of appropriate threshold values. Therefore, this section describes equations that enable to model the circuit operation. Equation 1 shows the relation of ACKs, ACKCs and NACKs monitored by FPM during a predefined quantity of flits (#flits).

$$\#\text{flits} = \text{ACKs} + \text{ACKCs} + \text{NACKs} \quad (1)$$

Aiming to explore ACKs, ACKCs and NACKs according to their probability of occurrence, we define Flit Ack Rate (FAR) and Flit Error Rate (FER) as the quantities of ACKs and ACKCs + NACKs captured inside #flits, respectively. Moreover, Equation 2 defines τ as a parameter that enables to capture the quantity of NACKs from the FER. Consequently, Equation 3 describes the quantity of ACKCs as the complementary probability defined in Equation 2.

$$\text{NACKs} = \tau \times \text{FER} \times \#\text{flits} \quad (2)$$

$$\text{ACKCs} = (1 - \tau) \times \text{FER} \times \#\text{flits} \quad (3)$$

Rewriting Equation 1 according to Equations 2 and 3 and the FAR definition, we provide Equation 4, which represents the probabilities of communications with and without faults.

$$1 = \text{FAR} + \tau \times \text{FER} + (1 - \tau) \times \text{FER} \quad (4)$$

FER and τ define probabilities of errors that depend on the technology of IC production, on the architectural design, and on the traffic behavior. Since traffic has dynamic behavior, it is hard to define its influence during the design time. On the other hand, the designer may define requirements of fault tolerance, and perform dynamic adjustment of thresholds to fulfill these requirements. We define FERmax as the maximum tolerated FER, and the threshold parameters are adjusted to fire fault tendency and permanent fault events, when this FER is reached.

FPM employs the counters NE, CE and DE to compute the quantity of ACKs, ACKCs and NACKs, respectively; and it employs N, M and P to define thresholds that enable to capture FERmax. The idea is to define an observation window based on the count of ACKs. During this observation window, it may capture fault tendency, or permanent fault, if counters CE, or DE reach M or P, respectively. On the other hand, if the counter NE reaches N, before firing fault

tendency or permanent fault events, means that eventual faults are considered transitory, and so they are discarded. Thus, the counters are reset and a new observation window starts. Figure 5 exemplifies a faultTend event based on the concept of observation window.

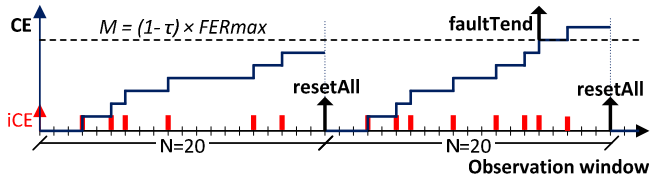

Figure 5: Example of fault tendency detection. The figure takes account only N and M thresholds. The input iCE represents instants of single fault events, which are computed by counter CE.

The procedure illustrated in Figure 5 shows that faultTend, resetAll, and even faultDet (not shown in the figure) are implemented as threshold relations, and the equations 5, 6 and 7 describe how the thresholds N, M and P are adjusted.

$$N = \#flits \quad (5)$$

$$M = (1- \tau) \times FERmax \quad (6)$$

$$P = \tau \times FERmax \quad (7)$$

4. Simulation Results

This work comprises two analyses: (i) the circuit behavior depending on the thresholds value and the size of observation window (defined by the quantity of flits passing through the monitored link); and (ii) the area and power consumption of FPM compared with a similar circuit presented in [6].

4.1 Threshold Settings for FPM

For faults occurrence evaluation, the signals are generated in SystemC and injected in the links during NoC simulation. All experiments employed 10% for double/simple error ratio.

In a first set of experimental analysis, we perform simulations observing a window variation from 150 to 40 flits, with 10,000 flits transmitted, considering counter NE with 8-bit length, and a FERmax of 20% (i.e. the FER used to define M and P, according to equations discussed in Section 3.1). The real FER varies from 5% to 50%. The main purpose of this simulation is to observe the FPM behavior, according to FERmax and τ parameters and sizes of observation windows, which depend on threshold N. Therefore, one can choose the values that better fit the target application. All results of this section are average values from 20 simulations. The confidence intervals are insignificant and kept hidden to provide cleaner figures.

Figure 6 shows that when the size of the observation window decreases, the FPM triggers less events of fault tendency. When the real FER is close to FERmax (i.e. the expected FER rate), the module can be well-adjusted, i.e. using the N setup to well-adjust the sensibility level of the FPM. When the real FER increases, it becomes impossible to set the adjust level as we can notice from 50% FER rate. This can be explained considering that higher values of N enables to capture more errors. Consequently, FPM becomes more sensitive.

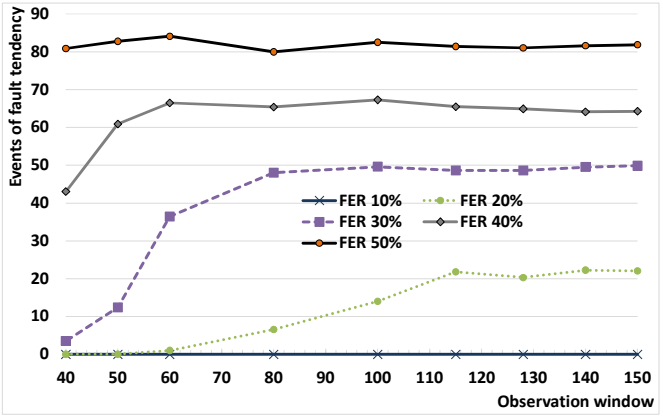

Figure 6: Occurrence of fault tendency events according to the size of the observation window (FERmax = 20%).

Figure 7 depicts that the behavior of FPM is similar to permanent fault events. For high FER rates, FPM detects much more events whereas the module is well-adjusted to 20% FER rate.

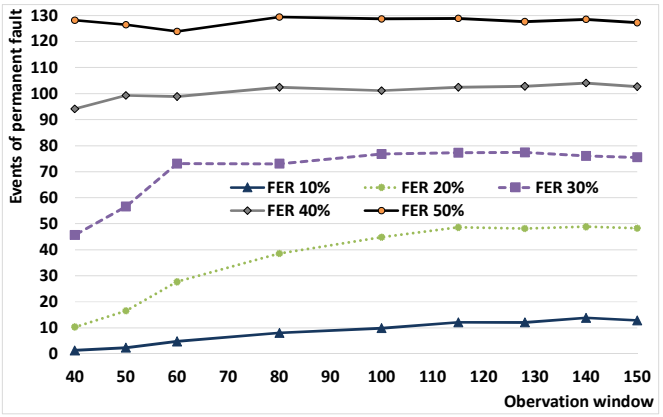

Figure 7: Permanent fault detection depending on the size of the observation window (FERmax = 20%).

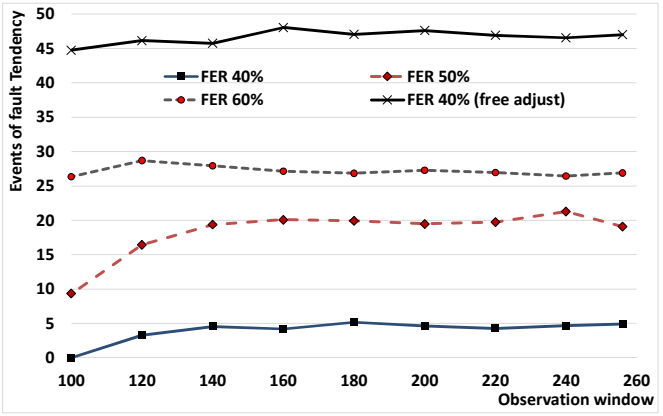

Figure 8: FPM with window size saturation and with free adjust of threshold settings.

Now, we are going to address the saturation problem for FPM (as the threshold counters are hardwired setup in project phase, the FPM can became insensible depending on relation between FERmax and real FER). Let suppose the length of

counter NE are hardwired to 8 bits, thus, the maximum counter size for N is 256. If the FERmax is adjusted to value 0.5, and the real FER is 0.4, the size of the observation window N becomes more sensible, once the value was saturated (refer to Figure 8). To address this problem we can use another rules to set M and P values. Herein, instead of using the equations of Section 3.1, we freely adjust M and P. In Figure 8, one can notice that the free adjust of M and P to a factor 2 activates the FPM again, even with a 40% FER rate (lower than FERmax, used for tuning setup phase).

4.2 Area and Power Analyzes

Table I presents the area and power reports of FPM and the circuit from previous work [6] in a standard cell implementation (UMC 90 nm library - very high-density 400,000 cell/mm², with decoupling capacity to limit glitches). FPM is more complex than the genuine one (logical cell, wires, registers, counter) thus it requires higher area and increased power.

According to detailed values in Table I, the genuine circuit size is about 5500 µm² and for the FPM is about 7570 µm². We conclude that the increasing of power consumption is about 60%, and the area used increases around 35%. The genuine circuit allows only detecting transient or non-transient faults. Our circuit permits to anticipate the tendency of these faults, although spending more resources.

Table I: Power and Area table.

	Requirement	Original		FPM	
Power (µW)	Internal	37.7		48.1	
	Switching	12.4		34.3	
	Total	58.1		82.4	
Area		cells	µm²	cells	µm²
	Combinational	1136	2840	1576	3940
	Non-combinational	1082	2705	1453	3633
	Buffer/Inv	25	63	24	60
	Total	2213	5608	3053	7633

5. Conclusion

The FPM circuit demonstrates to be a good solution to prevent potential faults during a transmission into a NoC, mainly considering its flexibility and potential on the parameterization of window size and threshold values. Our proposed circuit does not use a lot of extra area and power when compared with the genuine circuit presented in [6], mostly regarding the additional functionalities provided by FPM. The detection of fault tendency allows anticipating the transmissions behavior.

As future works, we intend to analyze the FPM behavior under different faults distribution, as Poisson or exponential.

6. Acknowledgements

This work is partially funded by FAPERGS (Docfix SPI n.2843-25.51/12-3 and PqG 12/1777-4) and CNPq-Brasil (process number 132778/2014-9).

7. References

[1] Ioannidis E. et al. “Evolution of Low Frequency Noise and Noise Variability through CMOS Bulk Technology Nodes from 0.5 µm down to 20 nm”. Solid-State Electronics, v. 95, pp. 28-31, May 2014.

[2] Benini L.; Micheli G. “Networks on chips: a new SoC paradigm”. Computer, v. 35, n. 1, Jan. 2002.

[3] Cuviello M.; Dey S.; Bai X.; Zhao Y. “Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects”. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 297-333, 1999.

[4] Marcon C. et al. “Phoenix NoC: A distributed fault tolerant architecture”. International Conference on Computer Design (ICCD), pp. 7-12, 2013.

[5] Aguiar, A. et al. “Hellfire: A design framework for critical embedded systems' applications”. International Symposium on Quality Electronic Design (ISQED), pp. 730-737, 2010.

[6] Dai L. et al. “Monitoring circuit based on threshold for fault-tolerant NoC”. Electronics Letters, v. 36, n. 14, July 2010.