Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 843239, 16 pages
doi:10.1155/2012/843239

Research Article

A New Efficient and Reliable Dynamically Reconfigurable *Network-on-Chip*

Cédric Killian, Camel Tanougast, Fabrice Monteiro, and Abbas Dandache

Laboratory of Interfaces, Sensors, and Microelectronics (LICM), University of Lorraine, EA 1776, 57070 Metz, France

Correspondence should be addressed to Camel Tanougast, camel.tanougast@univ-metz.fr

Received 24 February 2012; Accepted 9 June 2012

Academic Editor: Vivek Kumar Sehgal

We present a new reliable *Network-on-Chip* (*NoC*) suitable for *Dynamically Reconfigurable Multiprocessors on Chip* systems. The proposed *NoC* is based on routers performing online error detection of routing algorithm and data packet errors. Our work focuses on adaptive routing algorithms which allow to bypass faulty components or processor elements dynamically implemented inside the network. The proposed routing error detection mechanism allows to distinguish routing errors from bypasses of faulty components. The new router architecture is based on additional diagonal state indications and specific logic blocks allowing the reliable operation of the *NoC*. The main originality in the proposed *NoC* is that only the permanently faulty parts of the routers are disconnected. Therefore, our approach maintains a high run time throughput in the *NoC* without data packet loss thanks to a self-*loopback* mechanism inside each router.

1. Introduction

Nowadays, the trend for *Embedded Systems* is moving toward *Multiprocessor Systems on Chip* (*MPSoC*) in order to meet the requirements of real-time applications. The complexity of these *Systems on Chip* (*SoC*) is increasing and the communication medium is becoming a major issue in *MPSoC*. Generally, integrating a *Network-on-Chip* (*NoC*) in the *SoC* provides an effective way to interconnect several *Processor Elements* (*PEs*) or *Intellectual Properties* (*IPs*) (processors, memory controllers, etc.) [1]. The *NoC* medium features a high level of modularity, flexibility, and throughput. A *NoC* is constituted of routers and interconnections allowing the communications between the *PEs* and/or *IPs*. Communication on *NoC* relies on data packet exchanges. The paths for the data packets between a source and a destination through the routers are defined by the routing algorithm. Therefore, the paths that data packets are allowed to take in the network depend mainly on the adaptiveness permitted by the routing algorithm (partially or fully adaptive routing algorithm) which is locally applied in each router being crossed and for each data packet [2, 3].

Dynamically reconfigurable *2D* Mesh *NoCs* (*DyNoC, CuNoC, QNoC, ConoChi, etc.)* are suitable for FPGA-based systems [1, 4–7]. Indeed, thanks to partial dynamic reconfiguration of FPGAs [8], the number and the position of the *PEs or IPs* implemented on the FPGA Chip being dynamically modified during runtime allow more adaptiveness of the *MPSoC*.

To achieve a reconfigurable *NoC*, an efficient dynamic routing algorithm of data packets is required. The goal is to preserve flexibility and reliability while providing high *NoC* performances in term of throughput. Figure 1 illustrates a dynamic reliable *NoC*. Figure 1(a) presents the communications between several *IPs* whereas Figures 1(b) and 1(c) depict a dynamic placement of an *IP* or the occurrence of a faulty node where bypasses determined by dynamic routing algorithm are required. Furthermore, faulty nodes or even faulty regions (areas in the network having a size larger than a tile) also make the communications in the networks harder, and even impossible for some routing algorithms and required fault-tolerant algorithms, as shown in Figure 1(c). Therefore, dynamic component placements and faulty nodes or regions are the main reasons why fault-tolerant or adaptive algorithms have been introduced and used in run-time dynamic *NoCs* [4]. Regarding adaptive or fault tolerant routing algorithms, several solutions have been proposed

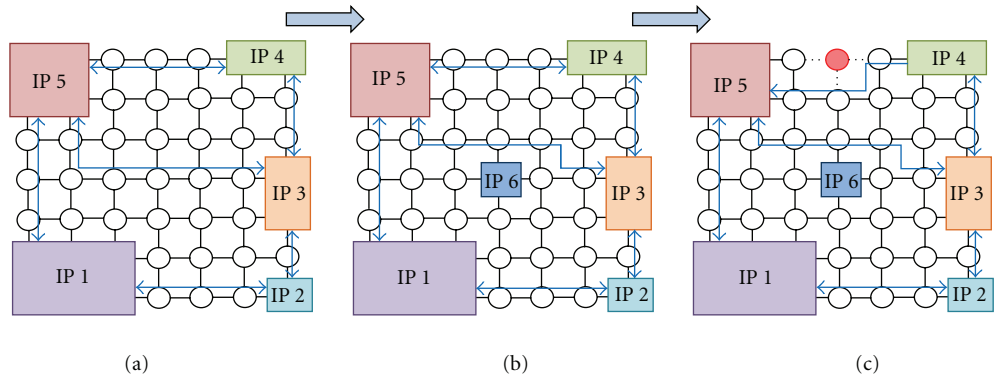

Figure 1: Illustration of a dynamic reliable *NoC*: normal operating (a), dynamic implementation of an *IP* (b), and on-line detection of a faulty router (c).

[9, 10]. Generally, these algorithms correspond to a modified *XY* routing algorithm allowing to bypass faulty or unavailable regions. In the case of adaptive routing algorithms based on the turn model [11], zones corresponding to already detected faulty nodes or unavailable regions in the *NoC* are defined. The neighbor routers of these zones must not send data packets toward these faulty routers or unavailable regions. To achieve that, chains or rings around the adjacent faulty nodes or regions are formed in order to delimit rectangular parts in the *NoC* covering all the faulty nodes or unavailable regions. In these chains or rings of switches, the routing tables are modified and differ from the standard tables related to the *XY* routing algorithm. These specific switches integrate in their tables additional routing rules allowing to bypass the faulty zones or regions dedicated to dynamic *IP*/*PE* instantiations, while avoiding *starvation*, *deadlock*, and *livelock* situations [11].

Regarding increasing complexity and the reliability evolution of *SoCs*, *MPSoCs* are becoming more sensitive to phenomena generating permanent, transient, or intermittent faults [12]. These faults can affect the data packet contents or generate routing errors. To detect these errors, specific error detection blocks are required in the network to locate the faulty sources. Moreover, permanent errors must be distinguished from transient errors. Indeed, permanent faulty parts of the *NoC* must be located precisely in order to be bypassed thanks to the adaptive routing algorithm. Consequently, the considered fault models used are *Stuck-at* faults for localization of permanent errors, and *Bit-flip* faults for transient errors [13].

To protect the data packets against errors, *Error Correcting Codes* (*ECC*) are implemented inside the *NoC* components [14, 15]. Among the well-known solutions, three are usually applied for the communications in *MPSoC* through a *NoC*. First, the *end-to-end* solution requires to implement an *ECC* in each input port of the *IPs* or *PEs* of *NoC* [14]. The main drawback of this solution is its incapacity to locate the faulty components (*PE*, *IP*, router, connection, etc.) in the *NoC*. Consequently, it is inadequate for dynamic *NoCs* where the faulty or unavailable zones must be bypassed. Second, the *hop-by-hop* detection is based on the implementation of *ECC* in each input ports of the *NoC* switches. For instance, in a router of four communication directions (*North*, *South*, *East*, *West*), four *ECC* blocks are implemented. Therefore, when a router receives a data packet from a neighbor, the *ECC* block analyzes the content of the packet to verify the correctness of the data. This process allows to detect and correct data errors according to the efficiency of the *ECC* being used. Third, another proposed solution is the *code disjoint* [14]. In this approach, routers include one *ECC* in each input and output data ports. This solution localizes the error sources which can be either in the switches or on the data links between routers. For all these techniques, each *ECC* implemented in the routers of the network requires an additional cost in terms of logic area, latency of the data packets and power consumption.

Concerning the routing error detections and among the existing techniques able to detect faulty routing decisions, we find the analysis of the source and destination addresses presented in [16, 17]. When a router receives a data packet, it compares its own address with the destination and source addresses. Then, the router checks its position in the deterministic *XY* path in the *NoC* of the considered data packet. The router performing this control is able to conclude if the switch from which it received the packet made or not a routing error according to the correct *XY* path. The drawback of this technique is the impossibility to handle the bypass of faulty nodes or regions. It results that this solution cannot be applied in adaptive or fault-tolerant routing algorithms. Indeed, as specified in a turn model algorithm [11], the structure of the reconfigurable *NoC* may contain bypass areas in which the switches must be able to take different routing decisions than the *XY* routing algorithm. To handle message routing errors in dynamic networks, a new faulty switch detection mechanism is required for adaptive or fault-tolerant routing algorithms. The aim is defining a new mechanism allowing the bypasses of the faulty nodes or regions (statically or dynamically *PEs*/*IPs* placed).

In this paper, we present a new reliable dynamic *NoC*. The proposed *NoC* is a mesh structure of routers which are based on a new architecture to detect the adaptive routing errors. Our approach includes data packet error

detections and corrections. The main originality of the proposed architecture is to localize precisely the error sources whereas using *Switch-to-Switch* error detections.

The remaining of this paper is organized as follows. Section 2 describes the architecture of the proposed reliable switch. Section 3 details the proposed routing error detection suitable for adaptive routing algorithms. Section 4 presents a specific self-*loopback* mechanism allowing to avoid losing data packets, maintain the performance of the *NoC,* and locate efficiency the permanent sources of data packet errors. Section 5 presents functional simulation validation while Section 6 gives several synthesis and performance results. Finally, conclusion and future works are given in Section 7.

2. Basic Concept of the *RKT-Switch*

We propose a new reliable *NoC*-based communication approach called *RKT-NoC*. The *RKT-NoC* represents a packet-switched network of intelligent independent reliable routers called *RKT-switch*. The architecture of the *RKT-switch* is depicted in Figure 2. The *RKT-switch* is characterized by an architecture of four directions (*North*, *South*, *East*, *West*) suitable for a 2D mesh *NoC*. The *PEs* or *IPs* can be connected directly to any side of a router. Therefore, there is no specific connection port for a *PE* or *IP*. Each port direction is composed of two unidirectional data bus (input and output ports). Each input port is associated with a fifo (buffers) and one routing logic block. The *RKT-switch* is based on *store-and-forward* switching technique. This technique is suitable for dynamically reconfigurable *NoCs*. Indeed, in our considered *NoC*, the *PEs* or *IPs* can be implemented by replacing one or several routers [6]. Therefore, by using the *store-and-forward* technique and when routers need to be reconfigured, just the emptying of the data packets stored in their buffers is required. Contrariwise, by using the *wormhole* switching technique [18], one data packet can be stored simultaneously in several routers. Consequently, the time required to empty all routers containing partial data packets and to reconstruct these packets before performing a reconfiguration is more significant.

The data flow control used in our architecture is the *Ack/Nack* solution which can handle fault tolerant transmissions and minimize the impact of the considered *Bit-flip* model while increasing the robustness against *SEU* [19]. This solution relies on the retransmission of the packets which are received as faulty by a neighbor node. To perform one retransmission after that a data packet is sent to a node, a copy of the packet is locally saved until a *Ack* or *Nack* is received. If a neighbor router receives a flit containing an error which cannot be corrected by the *ECC*, a *Nack* is sent and the whole packet is retransmitted. Otherwise, an *Ack* is generated at full packet reception. More precisely, an *Ack* can be generated only when the totality of the data packet has been received and checked by the router. This allows to reduce the latency. For our proposed *RKT-switch*, the *Hamming ECC* is considered in order to provide a tradeoff between area overhead and the data error correction capacity compared to others *ECC* [15]. This choice allows to correct *Single Event Upset* (*SEU*) errors (one error bit in a flit) and detect *Multiple Event Upset* (*MEU*) errors (two error bits in a flit). Moreover, the *Hamming code* is more suitable for *NoC* based on *Ack/Nack* flow control than the parity bit check. Indeed, in case of *Nack*, the data packet latency is increased due to the required retransmission. However, with the *Hamming ECC* a *SEU* detection does not require a retransmission and then the throughput is preserved. The distinction between the temporary and transient errors is obtained thanks to a *centralized journal*, saving the transmission results, and a *loopback output* mechanism (see Section 4). Furthermore, our solution combined with the *loopback* mechanism and the novel *centralized journal* allows to locate errors, either in the bus connections or inside the switches (more details in the Section 4). The main advantage of this mechanism is to limit the resources overhead due to the considered *Stuck-at* model. In addition, our reliable structure is based on *hop-by-hop* detection allowing the robustness against the *SEU* and two bits *MEU* errors while maintaining a good tradeoff between the area overhead and the capacity to locate errors. Moreover, the *IPs* connected in the *NoC* also require to integrate the proposed blocks for reliability mechanism. Indeed, these error detection techniques (routing error detection and error correcting code) allow online error detection on the boundaries of the *NoC*.

3. Routing Error Detection

The proposed reliable switch incorporates an online routing fault detection. This approach allows the routing error detections for adaptive algorithm based on the well-known *XY* routing algorithm. The main difficulty to detect the errors is to distinguish a bypass of an unavailable component in the *NoC* due to the use of adaptive algorithm from a real routing error due to a faulty component of the *NoC*. Figure 3 illustrates the challenge for such error detections. Apart from an increase of the data packet latency, the consequence of the nondetection of routing errors is the loss of data packets which can be sent either to an already detected faulty router or toward an area performing a dynamic reconfiguration. To achieve a routing error detection, the proposed reliable router relies on diagonal node state indications, additional routing information in the header flits, and *routing error detection* blocks in each port (see Figure 2). The basic concept of our approach is as follows: each router receiving a data packet checks the correctness of the routing decision made by the previous crossed switch.

3.1. Elements Required for the Routing Error Detections

3.1.1. Diagonal Availability Indications. RKT-switch uses information links to indicate to its neighbors its availability status. We define as unavailable an input port which cannot receive data packets. To preserve the highest throughput of the *NoC*, our strategy is to disconnect only the faulty parts of the routers. Thereby, if an input port router is permanently faulty, it is disabled while maintaining active

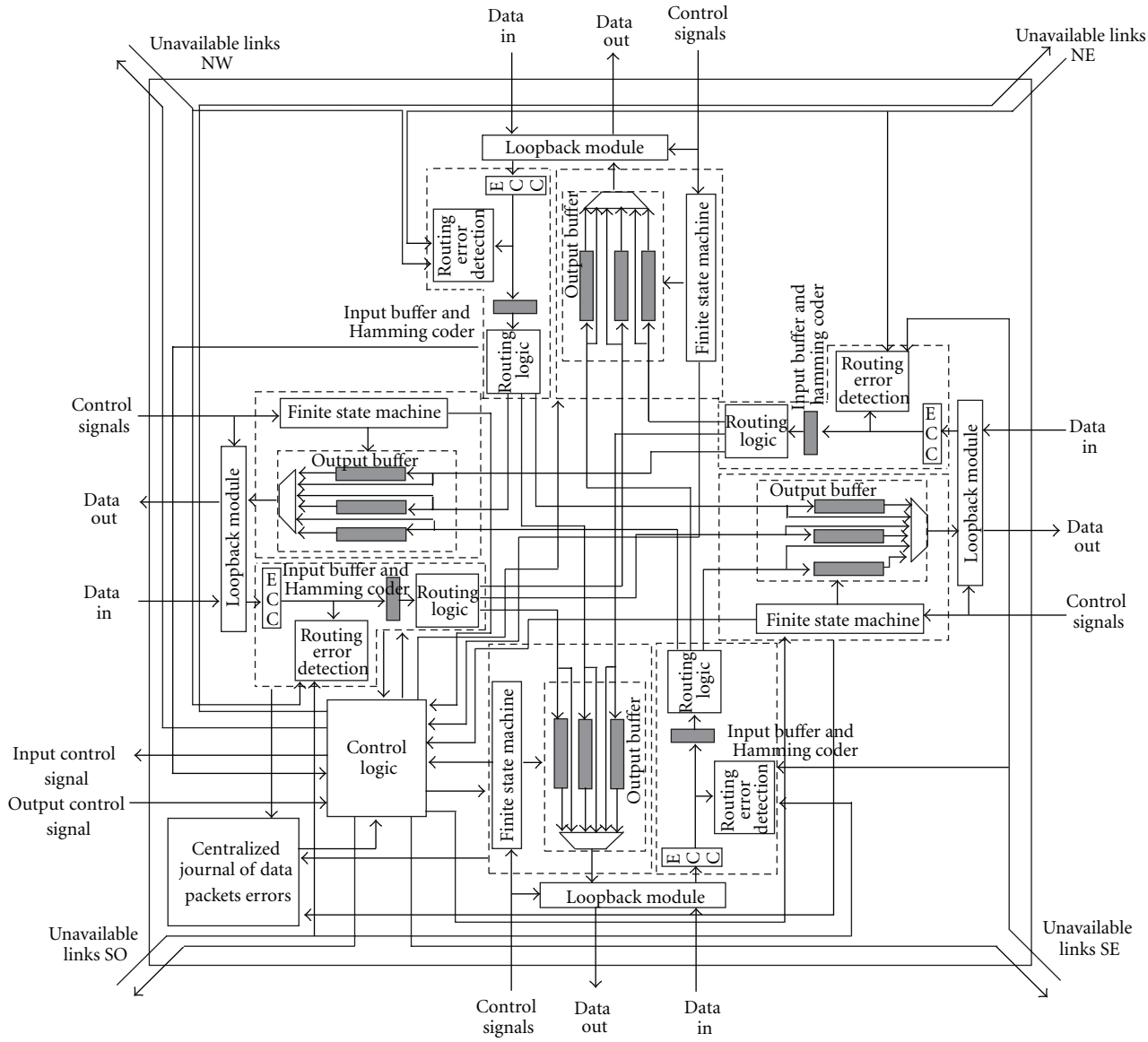

Figure 2: Architecture of the reliable router *RKT-switch*.

the others input ports in order to obtain a partial running switch. Contrariwise, if all input ports are faulty, the router is considered as unavailable. Similarly, we define as unavailable component, a component of the *NoC* which cannot receive data packets due to permanent faults or a partial dynamic reconfiguration.

The *RKT-Switch* indicates its availability status to the eight direct neighbor routers through the *Diagonal Availability Indication* (*DAI*) links. The network structure based on the *DAI* links is given in Figure 4. These *DAI* interconnections allow to check the correctness of the routing algorithm. Indeed, each router is able to control the availability status of the neighboring routers or components. For instance, in Figure 4, the router(i, j) can check the availability of the *North* and *South* input ports of the router$(i + 1, j + 1)$. The network components (*PEs* or *IPs*) are not allowed to route data packets and are limited only to accept data packets intended for them. Moreover, the *DAI* interconnections are set when components are instantiated in the neighborhood of the considered router.

3.1.2. Journal of Error Localizations. Each *routing error detection* block of the input routers, owns three register journals (*error* journals) to keep the routing error detection results. These journals correspond to the routing logic blocks of the neighbor router connected to the considered input port. For example in the Figure 4, the *West routing error detection* block of the router$(i + 1, j + 1)$ has three journals corresponding to the *West*, *North,* and *South* routing blocks of the router$(i, j + 1)$. Thanks to these *error* journals, the distinction between permanent and transient errors can be insured. In addition, from these journals, the location of the faulty routing algorithm blocks in the neighboring routers

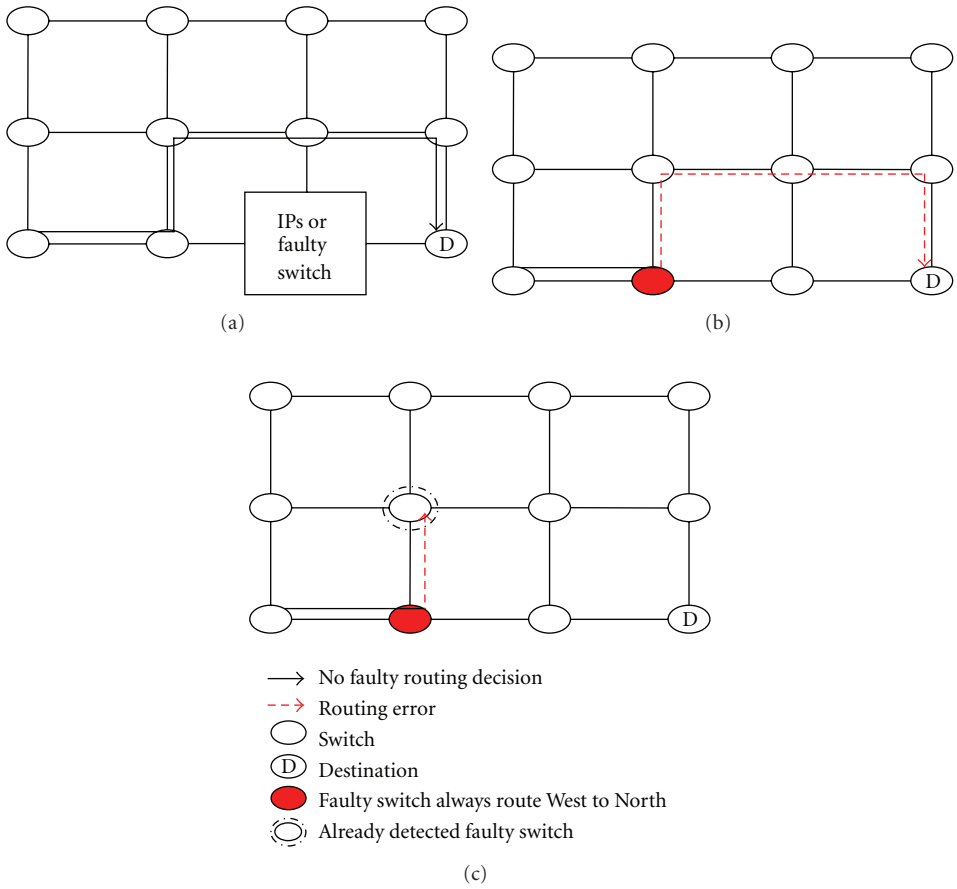

Figure 3: Illustration of the routing error detection problematic: to distinguish a dynamic bypass (a) from a routing error (b) and to avoid a loss of data packets (c).

can be deduced. Therefore, a permanent error is considered if three successive routing error detections occur for a specific *routing logic* block.

3.1.3. Structure of the Information Fields in the Data Packets. A *Sliding Gather Data* (*SGD*) field is added in each *header flit* of the transmitted data packets. Figure 5 details the structure of a data packet. A *Flit type* bit is used to distinguish the header from the data flits. The *SGD* field contains the addresses of the previous and penultimate routers crossed. Therefore, each router receiving a data packet checks the *SGD* field and validates the routing choice made by the previous router. To achieve the routing validation, the *SGD* field is updated at each router crossed during the data packet transmissions. This update is done by each *input buffer* block which also requires an update of *Hamming* code due to the modification of the header flit of the data packets. In addition, a *Unique Routing Path Indication* (*URPI*) bit is added to the header of the data packets. This bit is set if a local router has only one routing output path available. This bit allows to avoid false detections. Our approach allows one solution adapted for the dynamic *NoC*-based fault-tolerant routing algorithm.

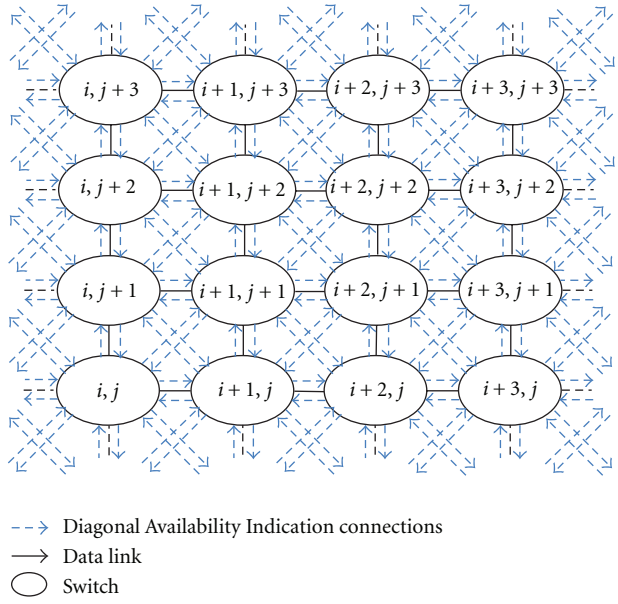

Figure 4: *Mesh*-based *NoC* with *Diagonal Availability Indication (DAI)* interconnections.

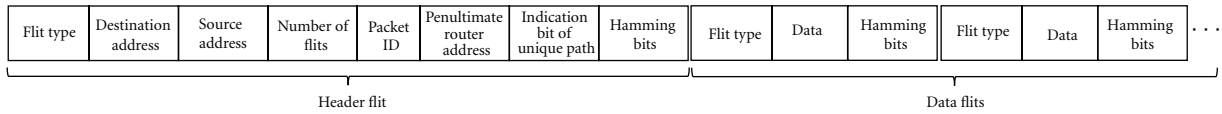

Figure 5: Structure of the data packets.

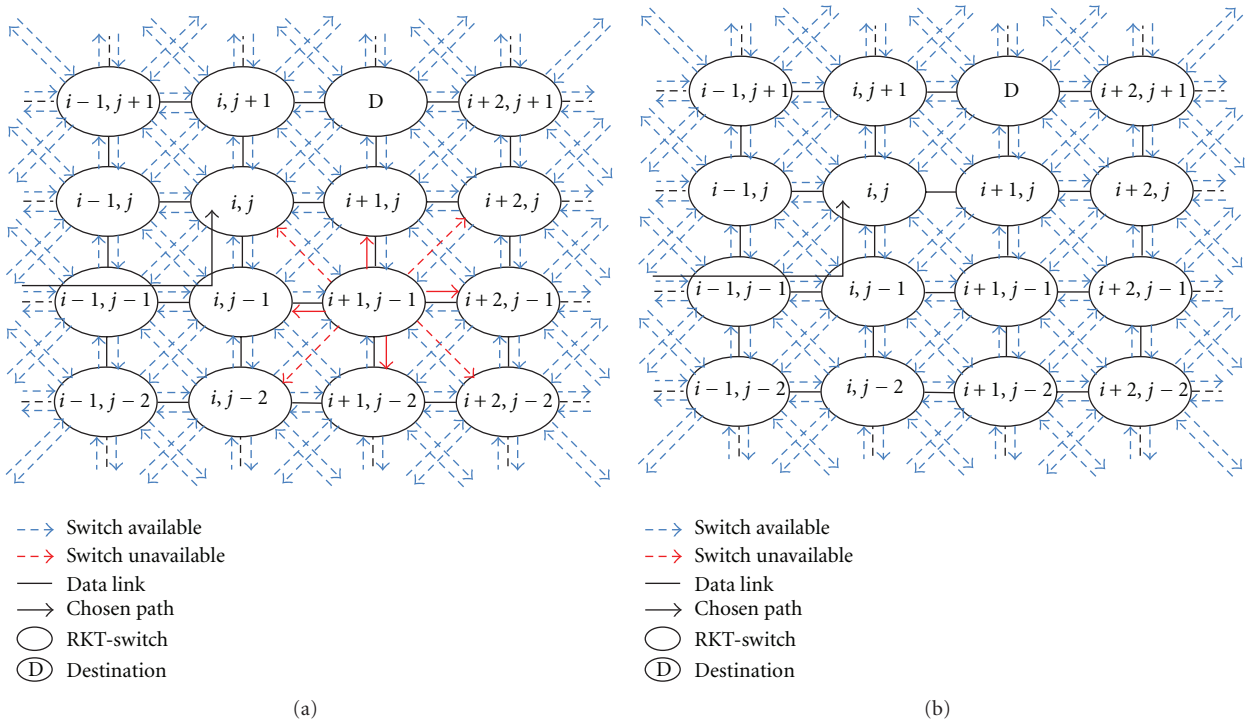

Figure 6: Case study of error detections: a dynamic bypass (a) and a routing fault (b).

3.2. Application of the Routing Error Detections. *RKT-switch* allows online routing fault detections and can distinguish routing errors from dynamic bypasses due to the adaptive routing algorithm. If a router is surrounded by three unavailable neighbors, it become also unavailable. Indeed, as we use nonbouncing routers [20], if a data packet is sent to a router surrounded by three unavailable nodes, the packet cannot be routed. Several application examples illustrate the efficiency of the proposed routing error detections.

Figure 6 gives two examples of data packets routed from the router$(i-1, j-1)$ toward the router$(i, j-1)$. In Figure 6(a), the router$(i, j-1)$ receives a data packet from its *West* neighbor router. It checks the correctness of the routing decision by the router$(i-1, j-1)$. In this case, the router$(i-1, j-1)$ respects the *XY* algorithm and no error is detected. In the Figure 6(b), the router(i, j) receives a data packet from the *South* direction. From the analysis of *SGD* field of this data packet and the penultimate router and destination addresses, the *XY* routing path between the router$(i, j-1)$ and the router$(i+1, j-1)$ is deduced. Consequently, the router(i, j) deduces that the router router$(i, j-1)$ do not respectes the *XY* routing algorithm. It results that the router(i, j) checks the *DAI* links of the router$(i+1, j-1)$. In the case where the router$(i+1, j-1)$ is unavailable, then the router$(i, j-1)$ makes a bypass decision and is not faulty (see Figure 6(a)). Otherwise, if the router$(i+1, j-1)$ is available, then the router$(i, j-1)$ makes a faulty routing decision (see Figure 6(b)). In Figure 6(b), the router(i, j) sets the *error* journal associated with the position of the *routing logic* block guilty of this routing error decision. This *routing logic* block is identified from the address of the penultimate node. More precisely, by checking the *SGD* field and identifying the penultimate router$(i-2, j-2)$, the *West routing logic* block of the router$(i-1, j-1)$ is deduced.

Figure 7 illustrates the role of the *URPI* bit. Here, the router(i, j) receives a data packet from the *North* direction according to the *XY* routing algorithm. By checking the *DAI* of the router$(i+1, j+1)$, a bypass decision is deduced. However, the destination is located on the *North* direction compared to the previous router. According to the routing algorithm used, the bypass should have been by the *North* direction. The availability of the router$(i, j+2)$ is deduced by checking the *URPI* in the *SGD* field. Indeed, if the *URPI* is activated, the router$(i, j+1)$ had only one remaining path and then did not made a routing error. Contrariwise, if the *URPI* is not activated, the router$(i, j+1)$ has two remaining available paths to route the data packets, and then it makes a routing error.

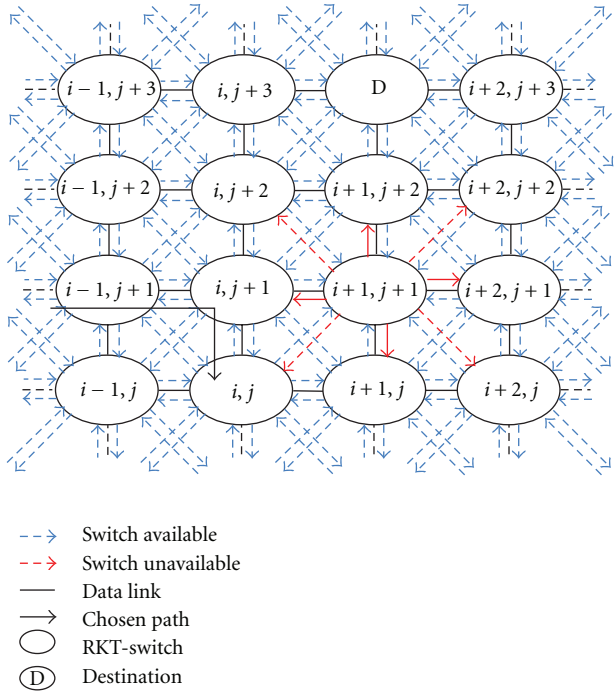

Figure 7: Case study of error detections: role of the *Unique Routing Parth Indicator* (*URPI*).

3.3. Principle of Routing Error Detection. *XY* based adaptive routing algorithms use the rules of the *XY*-algorithm [11] to route data packets into the network when there are available components. In case of an unavailable component, a specific routing path is locally chosen to bypass its position. When a router receives a data packet, it checks the correctness of the routing decision made by the previous node by applying the routing error detection algorithm depicted in Figure 8. If the address of the previous router (indicated in the *SGD* field) is the router itself, no routing error detection is performed due to a loopback of data packets (see Section 4). From the comparisons of addresses, the router checks if the previous routing decision has respected the *XY* routing algorithm. If it is respected, the previous decision is correct. Otherwise, the router defines if the previous decision is a bypass decision or a routing error. The detection algorithm requires to check the availability of the router in which the data packets should have pass according the *XY* algorithm. This control is performed thanks to the *DAI* links. If the router in the *XY* path is unavailable, the previous router makes a bypass decision. If it is available, the previous router makes a routing error. In this last case, the router adds a "1" in the error journal associated with the faulty *routing logic* block. The position of the faulty block is deduced from the address of the penultimate router in the *SGD* field. If three consecutive errors are performed by the faulty *routing logic* block, a permanent error is considered. In this case, a specific data packet is generated with as destination the faulty switch in order to inform of its failure. This specific data packet of one flit size indicates the faulty input port of the considered router that must be disconnected. The header flit is updated at each router crossed (addresses of the previous and penultimate routers). Consequently, the header flit requires to be coded by a *Hamming* coder. This block is implemented in the input buffer. By using a *NoC* based on data packets composed of several flits, it can happen to have one flit which is not following the header flit. In the proposed *RKT-NoC*, each flit has one bit indicating if the flit correspond at a header flit or a data flit as depicted in Figure 5. When a router receives the first flit of a data packet after the *Hamming* decoding, it checks if it is a header flit thanks to the *flit type* bit. If it is not a header flit, the flit is destroyed. Therefore, when the destination component receives a data packet, it counts the number of flits. If the number of the received flits does not match the number of flits indicated in the header flit, the packet is destroyed and a retransmission request is send to the emitter component.

3.4. Simulations. A 3×3 *RKT-NoC* with 3 *IPs* as illustrated in the Figure 9 has been simulated in *ModelSim* environment. The data packets contain 2 flits. In this simulation, we focus on the *routing error detection* block located in the *West* port of the router$(3, 2)$. The simulation results are shown in Figure 10. At the Event 1, the *error* journal corresponding to the *West* routing block of the neighbor router$(2, 2)$ already has one error detected. Similarly, the *North error* journal has detected two errors in the last two routing error detections. At Event 2, a header flit is analyzed. By checking the information in the header flit, we deduce the destination is the $IP(4, 1)$ and packet has been emitted by the $IP(2, 4)$. The previous router is the router$(2, 2)$ and the penultimate router is router$(2, 3)$. The data packet is received by the router$(3, 2)$ from the *West* direction input port. Next, the routing error detection algorithm is applied. As the *XY* routing path has not been respected, the availability of the router$(1, 1)$ is checked via the *diagonal_SW* signal. As this router is unavailable, the *North routing logic* block of the router$(2, 2)$ did not made a fault, and its corresponding *error* journal is set to "0." At Events 3 and 4, two data packets are received. Both are emitted from the router$(0, 2)$ and are destined to the router$(3, 4)$. For these data packets, the *XY* algorithm has not been respected. However, the router$(2, 3)$ which has transmitted these data packets is available. Consequently, a routing error is detected and a second "1" is added in the *West* error journal corresponding to the faulty *routing* block. The three previous routing error detections were positives, then a permanent routing error is deduced in the *West* direction input port of the router$(2, 2)$. This information is send to the control logic which sends a specific data packet to the neighbor in order to disconnect the faulty part of the router$(2, 2)$.

4. Loopback Block

In dynamic reconfigurable *NoC*, the components of the network can change during operation as illustrated in Figure 1. Indeed, the number and the position of the *PEs* and *IPs* in the *NoC* can be dynamically modified in order to meet the requirements of the application or to replace

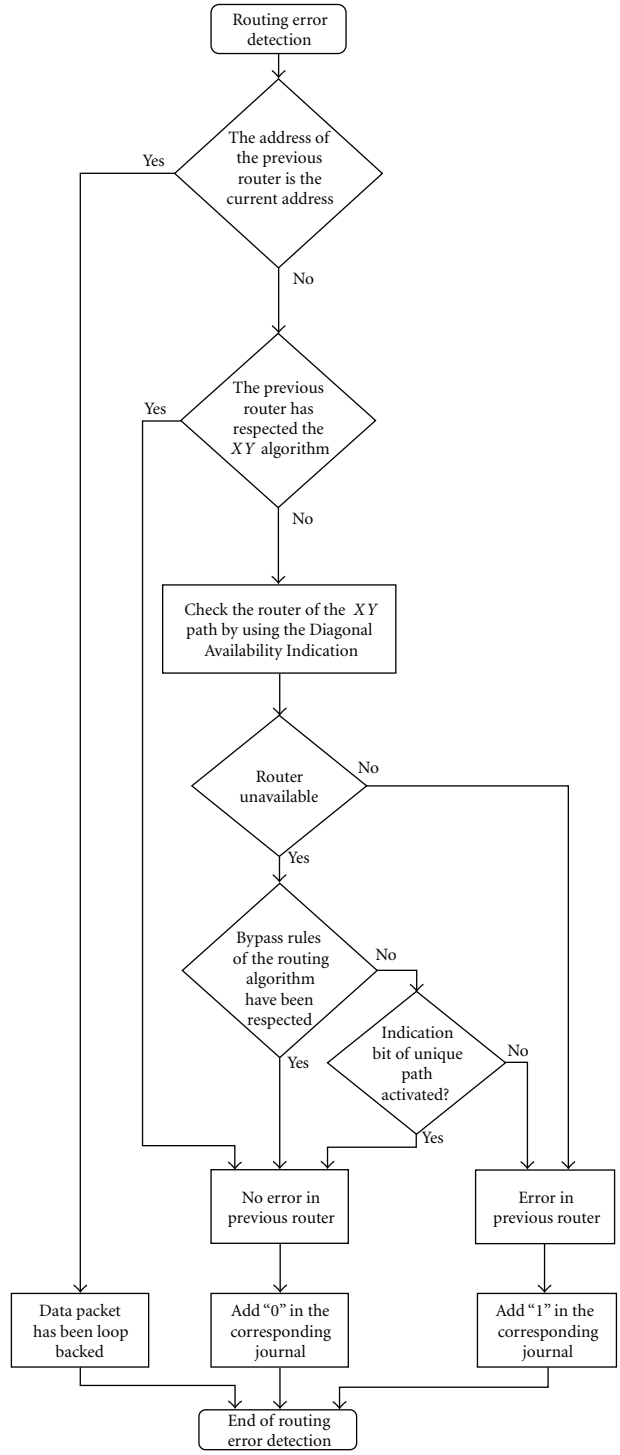

Figure 8: Routing error detection algorithm.

a detected faulty *IP*. To realize dynamic reconfiguration in the 2D *Mesh NoC* implemented in the FPGA, *Partial Reconfigurable Regions (PRRs)* have to be defined inside the FPGA [21]. These *PRRs* are regions where *Partial Reconfigurable Modules (PRMs)* are implemented. *PRMs* represent electronic circuits of functional units which are defined by specific partial-bitstreams and can be placed according to the application [21]. In practice, these *PRMs* correspond the *PEs* or *IPs* which are implemented and placed inside the dynamic *NoC*, as illustrated in Figure 1.

In a reliable *NoC*, faulty routers are run time isolated. If we consider a permanent faulty router which cannot be corrected, this router is permanently disabled and will never be used again. Similarly, during the reconfiguration of a *PRR*,

Table 1: Localization of the errors.

Result of the data transmission	Result of the data error detection after loopback	Input port	Output port	Data bus
Ack	No loopback required	No faulty	No faulty	No faulty
Three consecutive *Nack*	No error detected after loopback	No faulty	No faulty	Permanently faulty
Three consecutive *Nack*	Uncorrectable error detected after loopback	Suspicious	Suspicious	Suspicious

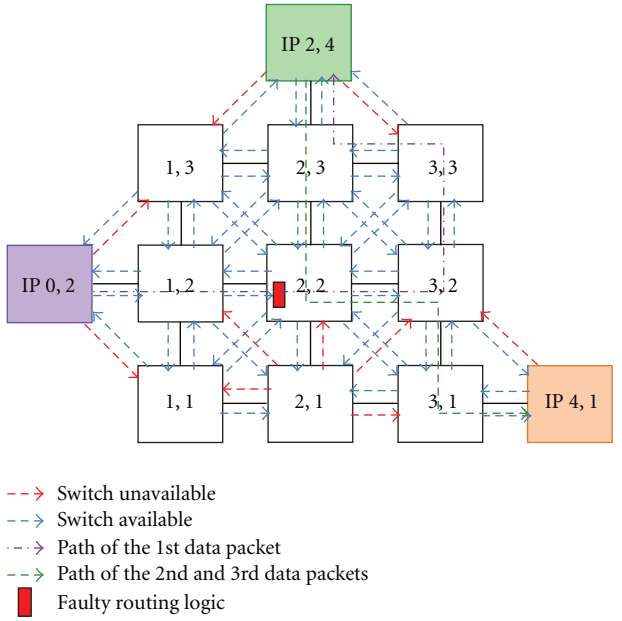

Figure 9: Illustration of the simulation results of the routing error detections in the *RKT-switch*.

no packet can be send inside the area being reconfigured. Thus, these active *PRRs* in the *NoC* are dynamically isolated. However, these isolations can lead to data packets losses and increase packet latency. These drawbacks occur when routers contain data packets in their output buffers while the neighboring nodes become unavailable due to dynamic implementations or permanent fault detections. Thereby, these data packets remain stored in the output routers. Consequently, data packets are either partially stored until the end of the reconfiguration (dynamic implementation case) or trapped and lost in the permanently faulty node detection case. To overcome these drawbacks, the proposed *RKT-switch* contains *output buffer* blocks associated with *loopback* blocks as described in Figure 2. The role of each *loopback* block is to empty the buffers of each output port by a looping back mechanism of data packets in the input port of the router (more details in the Section 4.3). It results that the looped back packets are re-routed toward another output port of the router and allow to avoid the data packets to be trapped. Figure 11 illustrates the role of a *loopback* block. A *PE* or *IP* emitter send toward a destination *IP* data packets according the *XY* routing algorithm. Suddenly, if we consider the router$(1,3)$ unavailable, the data packets remaining in the *West* output of the router$(2,3)$ are looped back and re-routed toward its *South* output. This mechanism allows then to route these stored data packets to the destination. Therefore, as the *router*$(1,3)$ is identified like unavailable, the follow data packets coming from the *East* input port are directly routed toward the *South* port by using the dynamic routing algorithm. Furthermore, one main advantage of using our proposed *loopback* block associated with a centralized journal of errors and the *switch-to-switch* data error detections (see Figure 2), is the precise location and distinction of the data error sources. Therefore, we can locate precisely if the data errors are on the data bus or in the input or output ports, and if the faults are permanent or transient.

4.1. Localization of Data Packet Errors. To locate the errors and to distinguish permanent and transient errors, a centralized journal of data packet errors is implemented as described in the Figure 2. This block is composed of journals corresponding to the input and output ports. These journals are shift registers with three bits of depth.

The *RKT-switch* uses the *Ack/Nack* data flow control. When a data packet is transmitted to a neighbor node, one copy of the data is stored until the acknowledgement is received. If no error occurred during the transmission (reception of an *Ack*), a "0" is added in the journal related to the input port of the get-in direction and the output port. If an uncorrectable error is detected by the neighbor, a retransmission is performed by generating a *Nack*. If the *Ack*/*Nack* data flow control of the considered router receives successively three *Nacks* from a neighbor, we consider a permanent error of the data packet sent by the considered router. Consequently, the data packet is looped back. Indeed, if three transmissions of one data packet failed, then the neighbor detects an uncorrectable error due to a permanent error. After the loopback, the data packet is checked by the input *ECC*. If no error is detected, we can conclude the errors were on the data bus. The error happened three consecutive times on the bus (i.e., three *Nacks*), then we can conclude a permanent error on the data bus. If an uncorrectable error is detected by the *ECC*, the error can be either on the bus, on the input port, or in the output port. The data packet is destroyed and a "1" is added in the input and output journals. Table 1 shows the correlation of the results of the data error detection allowing to deduce the position of the error (input or output block, or data bus), and if the error is permanent or transient. Indeed, if three consecutive errors occur in the same input or output journal, the *centralized journal* of data packet errors concludes a permanent error in the corresponding direction. The data packets being looped back after being checked by the *ECC* are controlled by the routing error detection block. However, this block finds in the *SGD* field that the previous router address is similar to its own address, deduces a loopback, and does not perform the routing error detection algorithm. When a permanent fault

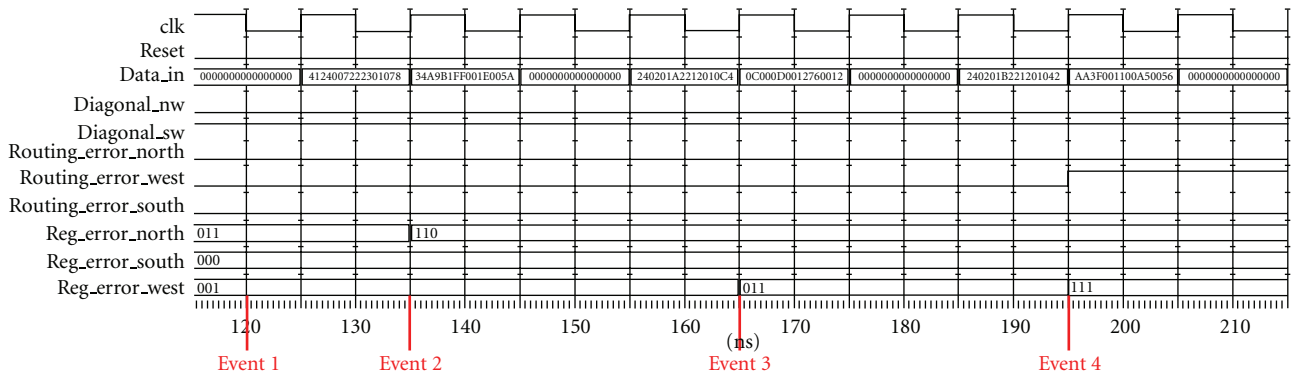

Figure 10: Simulation results of the routing error detections in the *RKT-Switch*.

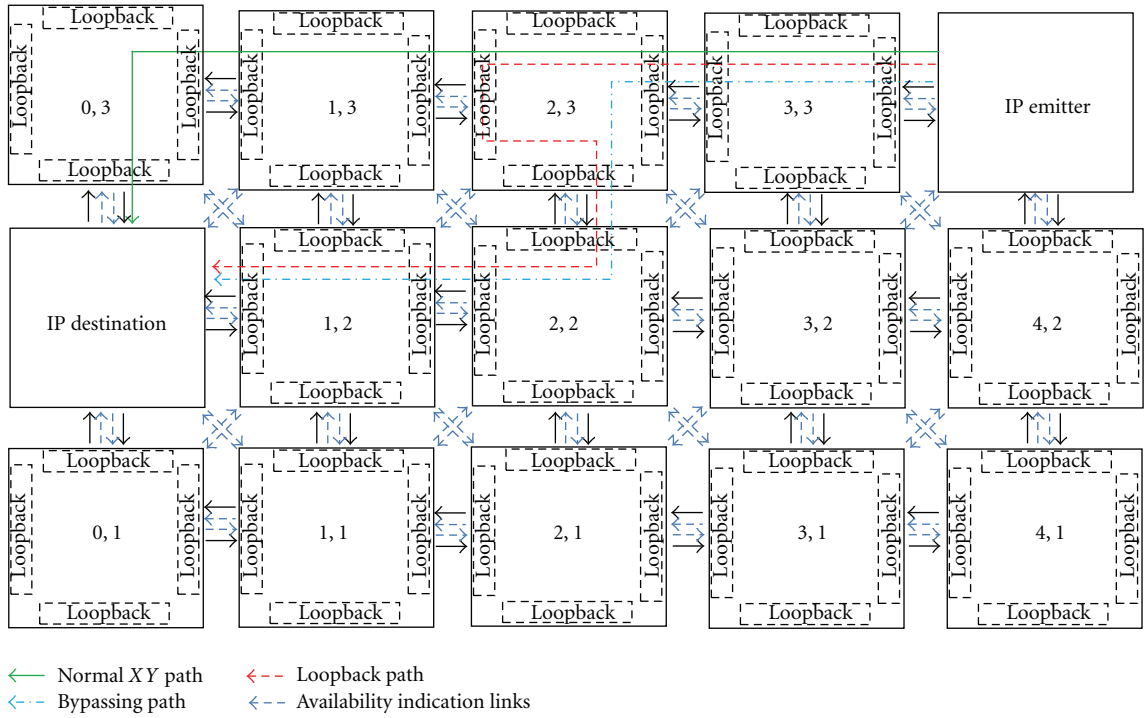

Figure 11: Illustrations of a data packet *loopback* and a dynamic bypass decision.

has been detected in a router, the faulty part of the *NoC* needs to be isolated. The part to isolate has been located precisely by using the journal and the loopback with the *switch-to-switch* error detections and can be either in the input port, output port, or data bus. If the error is in the input port, the router will locally activate the unavailable horizontal availability link of the faulty input port, and the two associated *DAI* links. In this way, the neighbor component connected to the faulty port will not send new data packet in this direction, and the *DAI* flags will indicate to the diagonal neighbors the possibility to bypass its position. If the error is on the data bus or in the output port, the router who detects the permanent error must indicate to the neighbor to activate its *DAI*. To indicate which port need to be disconnected, the router detecting the permanent fault send a data packet to the destination of the neighbor router. This data packet based on one flit contains the address of the destination router, the direction of the port to disconnect and the *Hamming* bits to protect the data. The router must not send this special flit in the direction that have been detected as faulty. Consequently, the data packet is generated in the input port corresponding to the direction of the faulty neighbor. As we consider *nonbouncing* routers, the data packets cannot be routed in the same direction as the get-in port. Therefore, the routing logic block will made a bypass routing and the packet will be send to the correct router.

4.2. Avoiding Data Packet Losses. The *RKT-switch* uses buffers in each output port as depicted in the Figure 2. This switch is suitable for *dynamic reliable NoC* in which

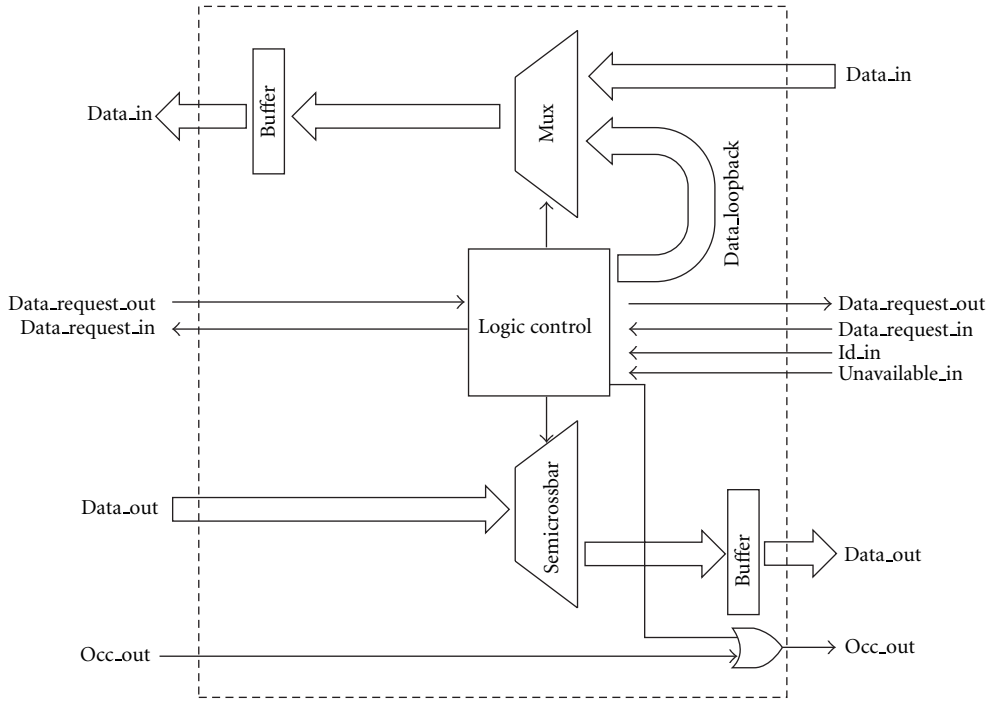

Figure 12: Architecture of the *loopback* block.

the components of the network can be modified in run time. If a *PRR* is dynamically reconfigured, no data packet can be sent to the reconfigurable area until the end of the reconfiguration process. Similarly, if a *NoC* router is becoming permanently faulty, the data packets need to bypass its position. However, if the output buffers are full while the neighbor router becomes unavailable (dynamic reconfiguration or permanent fault), the packets are trapped in the output port. In this case, the data packets are looped back inside the considered router in order to find a nonbusy or nonfaulty routing path. Therefore, the data packets are routed again toward a new output port of the switch, allowing to avoid the loss of the data packets. By emptying the output buffer, we reduce the data packet latency in the case where a neighbor router performing a partial reconfiguration. Similarly, the data packets are saved if the neighbor router is permanently unavailable. In the case of a loopback decision and emptying of the output buffer, the result of the *ECC* analysis is not used to disconnect the output port of the router. Indeed, the router require to distinguish a loopback due to an unavailable neighbor from a loopback request resulting of three consecutive *Nack* receptions. More precisely, if the *ECC* detects no error in the data packets after three consecutive *Nacks*, a permanent error is considered and located on the data bus.

4.3. Architecture of the Loopback Block. Figure 2 depicts the logic architecture of the *loopback* block. A *loopback output* block is implemented in the four ports of the router as illustrated in the Figure 2. The architecture of the *loopback output* block is depicted in Figure 12. The *logic control* block checks the availability of the neighbor router in order to transmit the data packets (*data_request_in* signal). If no loopback is required, one *semicrossbar* connects the buffer to the *data_out* signal in order to send the data packets toward the neighbouring router and activates the *data_request_out* signal. Next, a *multiplexor* connects the input data bus to the *data_in* bus. If a loopback is required in case where either a neighbor router is unavailable or an output block request occurs after three *Nack* receptions, the *logic control* block configures the *semi-crossbar* block to send the considered data packet on the *data_loopback* bus. Therefore, the data packet is looped back inside the router and will be considered as a new packet into the input port of the router. During this step and to avoid the reception of a new data packet from the neighbor switch, the *occ_out* signal is activated.

5. Simulation Results

To validate functionally the *RKT-NoC* communication approach, we simulated a 3×3 *NoC* connected to 12 communication modules. These simulations detailed the steps for the permanent error locations of the data packets based on the proposed approach. The *RKT-Switch* is configured to use data packets composed of 4 flits, and each input buffer can contain two data packets. Each communication module sends and receives data packets to the module connected to the opposite side of the network. Figure 13(a) illustrates the topology of the simulated network and the disposition of the modules.

Figure 14 is a snapshot of the simulation results. It can be seen at event 1 that all the modules send a data packet simultaneously. At event 2, the reception of all the data packets is also simultaneously. Indeed, all the data packets

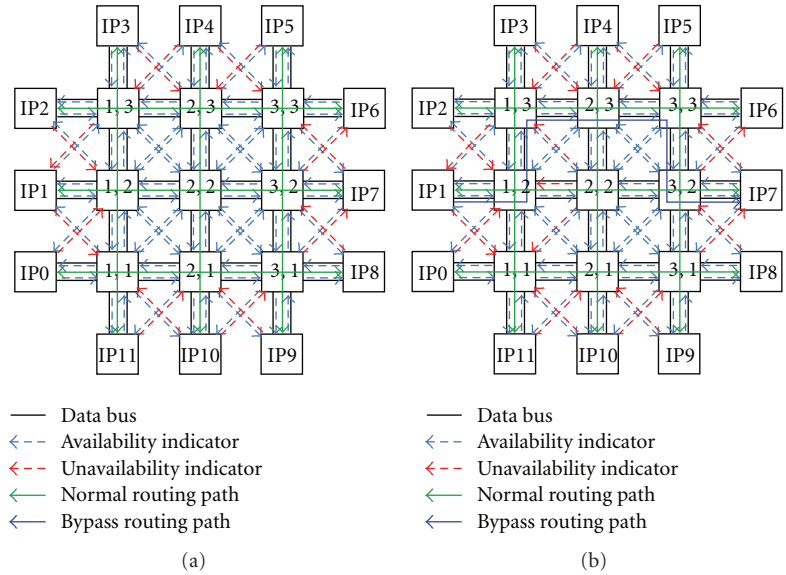

Figure 13: Simulations: (a) communication exchanges of 12 modules by a 3 × 3 *RKT-NoC*, (b) permanent error detected in a router's input.

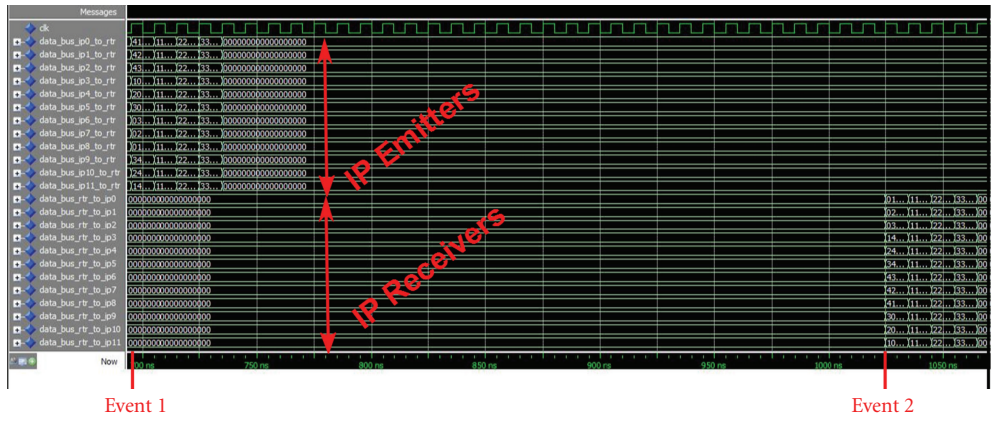

Figure 14: Simulation results: communication exchanges between 12 modules in a 3 × 3 *RKT-NoC*.

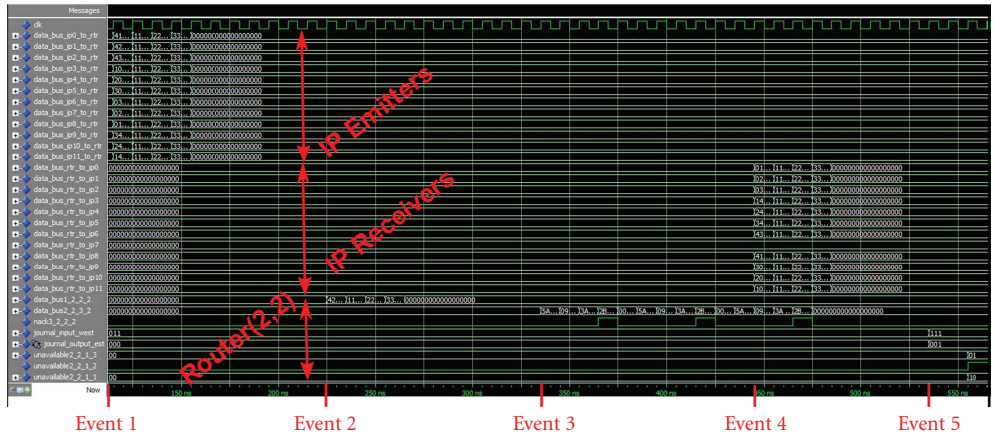

Figure 15: Simulation results: communication exchanges between 12 modules in a 3 × 3 *RKT-NoC* and detection of a faulty router input.

Figure 16: Simulation results: communication exchanges between 12 modules in a 3×3 *RKT-NoC* with a permanent fault detected in one router.

Table 2: *RKT-switch* synthesis results.

Data bus width (bits)	Virtex V			Virtex VI			Virtex VII		
	Slices Registers	Slices LUTs	f [MHz]	Slices Registers	Slices LUTs	f [MHz]	Slices Registers	Slices LUTs	f [MHz]
24	3503	4146	303.93	3537	5675	429.54	3540	5571	459.71
32	4308	4998	311.85	4377	6661	431.04	4340	6542	459.68
64	8402	8743	306.28	8360	11672	419.21	8362	11681	441.62

crossed the same number of routers, and they did not make a bypass.

The second simulation case represents a *RKT-NoC* in which an input block of the router(2, 2) is permanently faulty. The simulated fault is two bits stuck at "1" in the *West* input buffer. Figure 15 is a snapshot of the simulation results of the error detection. At event 1, all the modules are sending a data packet. Here, the centralized journal of the router(2, 2) already contains two errors for the west input. At event 2, the router(2, 2) receives a data packet in its *West* input. This packet is sent to the *East* neighbor at event 3. The neighbor router(3,2) generates three consecutive *Nacks* due to an error which cannot be corrected by the *Hamming ECC*. The data packet is looped back, and the result of the error detection performed in the *ECC* of the *East* input is positive. This result is stored in the centralized journal at event 5. The three last results for the *West* input were positive. The router concludes to a permanent error and activates its unavailable flags to the *West*, *North-West,* and *South-West*. All the data packets have been received at event 4 excepted one.

The next part of the simulation is presented in the Figure 16. At event 1, all the modules send one data packet. At event 2, all the data packets are received excepted one which is received at event 3. This is due to a bypass of the router(2, 2) for the data packet sent from the *IP*1. Indeed, this data packet crossed two more routers increasing the latency as illustrated in Figure 13(b).

These simulation results show one of the main advantages of the *RKT-NoC* which is the possibility to detect on run-time permanent errors, and to disconnect partially the faulty blocks of the switches. In these simulations, only one input port of the router(2, 2) has been disconnected from the network. This allows to keep as high as possible the number of available paths, and then to maintain the throughput.

6. Synthesis Results and Performance Evaluation

6.1. FPGA Synthesis Results. The results are obtained by considering of *RKT-Switches* configured to data packets of 4 flits and where each input buffer can contain two data packets. Table 2 gives the synthesis results in terms of *Slices Registers*, *Slices LUTs* and *maximal working frequency* for different data bus size and FPGA technology (Virtex V, Virtex VI, and Virtex VII Xilinx FPGA). We note that the 32-bits *RKT-Switch* requires 4340 *Registers*, 6542 *LUTs* and can operate up to 459.6 MHz on the Virtex VII FPGA technology. The operating frequencies decrease depending of the occupancy rate of the implementation design in the FPGA chip. Table 4 gives the logic resources required for each block of the proposed reliability concept of the router and depending to the data bus size. These blocks can be directly used in others *NoC* routers based on *XY* routing algorithms in order to integrate some parts of the proposed mechanism by taking into account the area overhead. We observe that the main area overhead is obtained for the *output buffer* block corresponding ot 46% of the router in Xilinx Virtex VI FPGA. We have also synthesized several *RKT-NoC* sizes in Xilinx Virtex VI technology. These results are given in the Table 3 in terms of *Slices Registers* and *Slices LUTs*. Moreover, an estimation of the power consumption is given through the Xilinx *XPower Estimator* tool [22]. The results demonstrate that our proposed architecture can be efficiency implemented with FPGA technology. It can be stated that an attractive tradeoff between high speed and

Table 3: *RKT-NoC* synthesis results.

Data bus width (bits)	*RKT-NoC*4 × 4			*RKT-NoC*6 × 6			*RKT-NoC*7 × 7		
	Slices registers	Slices LUTs	Power (W)	Slices registers	Slices LUTs	Power (W)	Slices registers	Slices LUTs	Power (W)
24	56840	95279	1.929	127900	199471	3.450	171300	256759	4.315
32	68352	104556	2.034	153165	223147	4.679	207973	304019	5.316
64	133920	201074	3.448	301650	431531	6.969	410590	587364	8.215

Table 4: Synthesis results of the *RKT-switch*'s blocks-area ratio.

Name of blocks	Number of each block	64 bits				32 bits			
		Number of slices		% of the switch		Number of slices		% of the switch	
		Registers	LUTs	Registers	LUTs	Registers	LUTs	Registers	LUTs
Input block	4	662	1216	31.4	39.3	367	688	30.3	37.4
Output block	4	988	1275	46.9	41.2	558	774	46	42.1
Loopback	4	159	187	7.6	6	93	121	7.7	6.6
Routing error detection	4	39	46	1.9	1.5	24	31	2	1.7
Centralized Journal	1	32	61	0.4	0.5	32	61	0.7	0.8
Control logic	1	32	103	0.4	0.8	32	103	0.7	1.4
Hamming coder	4	72	49	3.4	1.6	31	22	2.6	1.2
Hamming decoder	4	166	239	7.9	7.7	99	130	8.2	7.1
Routing logic	4	3	41	0.1	1.3	25	32	2.1	1.7

low logic resources has been achieved. Indeed, our 4 × 4*NoC* implementation on a Xilinx Virtex-VI device uses only 201074 *Slices-LUTs* for a 64 bits bus size. We note from the results in the Table 3 that the power consumption overhead generated by the proposed detection and diagnosis mechanism is limited and suitable for the FPGA implementation.

6.2. Performance Evaluation

6.2.1. Flit Injection Rate. The *Packet Injection Rate (PIR)* is the number of data packets that can be send in one clock cycle. For instance, if an *IP* has a *PIR* of 0.5, that means it can send 50 data packets in 100 clock cycles. The *Flit Injection Rate* (*FIR*) is the value of the *PIR* multiplied by the number of flits in each data packet. We have estimated the $\text{FIR}_{\max}$ by simulating different sizes of *NoC*: 1 × 1, 3 × 3 and 4 × 4. Each *RKT-NoC* is connected to the maximum number of communication modules. The $\text{FIR}_{\max}$ is obtained when the network is working without unavailable component, and when the modules send and receive data packets only to the neighbor located at the opposite side of the network. Indeed, by sending packets with this traffic, no case of congested router can occur. The $\text{FIR}_{\max}$ has been estimated to 0.369 for any *RKT-NoC* size and any number of *IPs*.

6.2.2. Latency. In a $n \times n$ *RKT-NoC*, the minimal latency ($\text{latency}_{\min}$) to cross the network from source to destination is defined by (1). N_{RKT} and $\text{latency}_{RKT\,\min}$ are the number of switches crossed and the minimal latency to cross one switch, respectively. The following equation takes into account the additional clock cycles of the *Ack/Nack* data flow control used for our reliable switches. This data flow control technique requires two clock cycles latency per router transmission:

$$\text{Latency}_{\min} = N_{RKT} * \text{latency}_{RKT\,\min} + N_{RKT} * 2. \quad (1)$$

In real-time applications, the data packet latency depends on the traffic network. We have evaluated the average latency for different sizes of *RKT-NoC*: 1 × 1, 3 × 3 and 4 × 4. To evaluate the latency, we have simulated these *NoCs* surrounded by the maximum number of communication modules: 4 modules for the 1 × 1 *NoC*, 12 modules for the 3 × 3 *NoC*, and 16 modules for the 4 × 4 *NoC*. Each module sends and receives data packets. The destinations of the data packets are generated randomly. The emission of data packets are performed at the maximum *PIR*. Table 5 gives the minimal and maximal average latencies for each *NoC* size in terms of clock cycles and timing (ns) by considering the maximum working frequency obtained with Xilinx Virtex V, VI, and VII FPGAs. It results that, for a 3 × 3 *RKT-NoC*, the minimum and maximum average latencies are 73 and 129 clock cycles, respectively.

6.2.3. Throughput. The maximum throughput of a *RKT-Switch* depends on the data bus width of n bits, the working frequency f, and the *FIR*. The $\text{throughput}_{\max}$ is given by (2) by considering the maximum number of connected *IPs* of 4:

$$\text{Throughput}_{\max} = 4 * n * \text{FIR}_{\max} * f. \quad (2)$$

If we consider a data bus size of 64 bits in which 4 *IPs* are connected, and the maximum operating frequency of the *RKT-Switch* in the Virtex VII FPGA technology,

Table 5: Average latency evaluation of 10 000 packets sent per communication module with a random traffic and a maximum PIR.

Latency	*RKT-NoC* size					
	1 × 1		3 × 3		4 × 4	
	Min	Max	Min	Max	Min	Max
Clock cycles (nb)	20.86	21.60	73.10	129.43	96.33	168.66
Virtex VII (ns)	47.24	48.91	165.53	293.09	213.59	381.91
Virtex VI (ns)	49.77	51.52	174.38	308.75	225.01	402.32
Virtex V (ns)	68.12	70.52	238.68	422.59	307.98	550.66

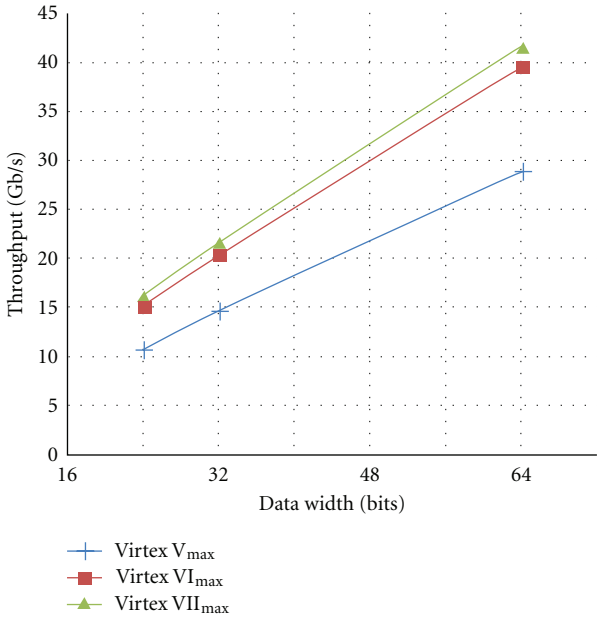

Figure 17: Throughput of one *RKT-Switch* for different data widths.

the Throughput$_{max}$ is 41.72 Gbps. Figure 17 gives the Throughput$_{max}$ of one *RKT-Switch* for different data bus widths and operating frequencies. The theoretical maximum throughput of one *RKT-NoC* is defined by the Throughput$_{max}$ of one router multiplied by the number of switches in the *NoC*.

6.3. Robustness Evaluation. To evaluate the robustness of the system against *Single Event Upsets* (*SEUs*), we have simulated the synthesizable architecture of a 6 × 6 *RKT-NoC* surrounded by 24 *IPs* generating random-traffic. This simulation has been realized for 3 different *FIR*: 0.13, 0.2 and maximum rate. For each *FIR*, we have injected *SEUs* at different frequency as specified in the Table 6. For instance, the *frequency* 15 means one *SEU* is injected in each 15 clock cycles. The position of the *SEU* in the network is random. For each configuration, 2.4 billion data packets have been transmitted. Table 6 shows the number of packet lost for the different simulation configurations. We can see that for the maximum *FIR* and a *SEU* frequency of 25, only 8 data packets are lost. We can clearly show that the number of packet lost increases with the *FIR*. Indeed, the number of data packets waiting in buffers when a neighbor is busy increases with the *FIR*. Thus, when data packets are waiting in a buffer, we have a probability to have two consecutive *SEUs* on the same data packet. We use the *Hamming* correcting code which can only correct one bit error and detect two errors. If a data packet receives one *SEU* in an input or output buffer and another one on the data bus, the retransmission protocol (i.e., *Ack*/*Nack*) saves the data packet. However, if a second *SEU* occurs also in input or output buffer, the data packet is lost. The same tendency is observed when increasing the *SEU* frequency or/and the *FIR* (see the Table 6).

Table 6: Number of data packets lost for several error injection frequency and Flit Injection Rates.

Flit Injection Rate	SEU frequency				
	50	25	15	10	5
0.14	0	1	0	2	4
0.2	2	1	8	14	48
Maximum	7	8	18	64	194

7. Conclusion

In this paper, we have proposed a new reliable switch architecture for *2D* mesh *NoCs*. This reliable architecture integrates a new online routing detection suitable for dynamic routing algorithms. Our detection approach allows to distinguish a routing error from a routing bypass decision of faulty nodes in the network and locates the specific faulty routing block. The *MPSoC* data packets are protected by using *Error Correcting Code* (*ECC*) of *Hamming* and a *switch-to-switch* error detection. Furthermore, the proposed *loopback* block in each output ports of the proposed switch allows to distinguish and locate precisely the data error sources (input or output ports, data bus between nodes, etc.). One originality of our solution is that with proposed reliable *NoC*, only the faulty parts of the switches are disconnected allowing to maintain online the throughput of the *NoC*. The FPGA synthesis of the proposed reliable *NoC* shows attractive performances since it leads to design with small logic area, satisfactory throughput rates, and low latency while providing an efficient online errors detection suitable for *NoC*-based reconfigurable systems.

References

[1] J. Shen and P. Hsiung, *Dynamic Reconfigurable Network-on-Chip Design: Innovations For Computational Processing and Communication*, IGI Global, 2010.

[2] G. M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, 2000.

[3] Y. M. Boura and C. R. Das, "Efficient fully adaptive wormhole routing in n-dimensional meshes," in *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pp. 589–596, IEEE, June 1994.

[4] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "DyNoC: a dynamic infrastructure for communication in dynamically reconfigurable devices," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 153–158, IEEE, August 2005.

[5] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–6, August 2006.

[6] S. Jovanović, C. Tanougast, and S. Weber, "A new high-performance scalable dynamic interconnection for FPGA-based reconfigurable systems," in *Proceedings of the IEEE 19th International Conference on Application-Specific Systems, Architectures and Processors (ASAP '08)*, pp. 61–66, July 2008.

[7] S. Jovanović, C. Tanougast, C. Bobda, and S. Weber, "CuNoC: a dynamic scalable communication structure for dynamically reconfigurable FPGAs," *Microprocessors and Microsystems*, vol. 33, no. 1, pp. 24–36, 2009.

[8] P. Lysaght and J. Dunlop, "Dynamic reconfiguration of fpgas," in *Proceedings of the Selected Papers from the Oxford 1993 International Workshop on Field Programmable Logic and Applications on More FPGAs*, pp. 82–94, Abingdon EE&CS Books, 1994.

[9] J. Wu, "A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1154–1169, 2003.

[10] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant network-on-chip architectures," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN '06)*, pp. 93–104, IEEE, June 2006.

[11] S. Jovanović, C. Tanougast, S. Weber, and C. Bobda, "A new deadlock-free fault-tolerant routing algorithm for noc interconnections," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 326–331, September 2009.

[12] C. Grecu, L. Anghel, P. P. Pandea, A. Ivanov, and R. Salen, "Essential fault-tolerance metrics for NoC infrastructures," in *Proceedings of the 13th IEEE International On-Line Testing Symposium (IOLTS '07)*, pp. 37–42, July 2007.

[13] J. Borecky, M. Kohlik, P. Kubalik, and H. Kubátová, "Fault models usability study for on-line tested fpga," in *Proceedings of the 14th Euromicro Conference on Digital System Design (DSD '11)*, pp. 287–290, IEEE Computer Society, Washington, DC, USA, 2011.

[14] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, and P. P. Pande, "On-line fault detection and location for NoC interconnects," in *Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS '06)*, pp. 145–150, July 2006.

[15] H. Zimmer and A. Jantsch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES '03)*, pp. 188–193, ACM, New York, NY, USA, October 2003.

[16] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi, "Online NoC switch fault detection and diagnosis using a high level fault model," in *Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'07)*, pp. 21–29, IEEE, 2007.

[17] N. Karimi, A. Alaghi, M. Sedghi, and Z. Navabi, "Online Network-on-Chip switch fault detection and diagnosis using functional switch faults," *Journal of Universal Computer Science*, vol. 14, no. 22, pp. 3716–3736, 2008.

[18] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[19] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in *Proceedings of the 18th Symposium on Integrated Circuits and Systems Design (SBCCI '05)*, pp. 224–229, September 2005.

[20] J. Raik, V. Govind, and R. Ubar, "An external test approach for network-on-a-chip switches," in *Proceedings of the 15th Asian Test Symposium*, pp. 437–442, IEEE, November 2006.

[21] Xilinx, "Early access partial reconfiguration user guide," in *Xilinx*, 2008.

[22] Xilinx, "Xilinx power tools tutorial," in *Xilinx*, 2011.