Secure and Dependable NoC-Connected Systems on an FPGA Chip

Taimour Wehbe, *Student Member, IEEE*, and Xiaofang Wang, *Member, IEEE*

***Abstract*—The growing reliance on intellectual properties exposes systems on chip (SoCs) to many security vulnerabilities and is raising more and more concerns. At the same time, with the quick increase in chip density and deep scaling of feature size, current billion-transistor chip designs introduce more challenges to manufacturing fault-free chips. In this paper, we propose an integrated run-time solution for both security and fault tolerance of field-programmable gate arrays (FPGA)-based SoCs through digital signatures, live monitoring, adaptive routing, and partial reconfiguration supported by an in-house-developed network on chip, X-Network. Our X-Network reduces the ratio of required routers versus processing elements with better performance, and more importantly, offers more flexibility than conventional networks to facilitate fault tolerance and security designs. A multiplexed Montgomery modular multiplication architecture is used to increase the speed of the Rivest-Shamir-Adleman algorithm. The design has been implemented and tested on a Xilinx Virtex-6 FPGA development board. Experimental results show that even if up to 20% of the links in the network are faulty, our reconfigurable architecture and algorithm manage to route packets around such faults, and deliver them to their destinations in a relatively low latency. Unlike conventional fault-tolerant methods that usually require resource redundancy, our design does not incur significant area or speed degradation. The resource overhead for both security and fault-tolerant features is around 10% more lookup tables.**

***Index Terms*—Fault tolerance, field-programmable gate arrays (FPGA), hardware security, network on chip (NoC), partial reconfiguration.**

Nomenclature

CMOS	Complementary metal–oxide–semiconductor.
IP	Intellectual property.
FPGA	Field-Programmable gate array.
ASIC	Application-specific integrated circuit.
3PIP	Third-party intellectual property.
AES	Advanced encryption standard.
SoC	System-on-chip.
NoC	Network-on-chip.
PE	Processing element.
DSP	Digital signal processing.
CRT	Chinese remainder theorem.
LUT	Lookup table.
IC	Integrated circuit.
BRAM	Block RAM.
MUX	Multiplexer.
DeMUX	Demultiplexer.
CIA	Confidentiality, integrity, authentication.
MAC	Message authentication code.
VC	Virtual channel.
IOB	Input/output block.
RM	Reconfigurable module.
RP	Reconfigurable partition.
XPS	Xilinx platform studio.
SDK	Software development kit.
SL	Static logic.
CF	Compact flash.
RSA	Rivest-Shamir-Adleman.

Manuscript received October 23, 2015; revised March 14, 2016 and June 6, 2016; accepted August 23, 2016. Associate Editor: W.-T. Chien.

T. Wehbe was with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA. He is now with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: taimour.wehbe@gatech.edu).

X. Wang is with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA (e-mail: xwang@villanova.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2016.2606883

I. Introduction

STATE-OF-THE-ART system-on-chip (SoC) field-programmable gate arrays (FPGAs) integrate multicore advanced RISC machine (ARM) processors, many commonly needed peripherals, and memory blocks with the reconfigurable logic on a single chip, offering many advantages to system designers [1]. The dramatic increase in complexity and magnitude of the system design has made it necessary to reuse design and to employ intellectual properties (IPs) from different vendors, including those from overseas third parties. This raises serious concerns about security and trustworthiness of the products employed in critical applications like military, health, aircrafts, etc. [2]. Malicious logic hidden in an IP core might be designed to disable or destroy a system, or to leak confidential information. When multiple IP cores interact with each other at the chip level, the probability of them being exploited becomes higher. The authors show in [3] that a backdoor on a military-grade FPGA chip allows the attacker "to extract all the configuration data from the chip, reprogram crypto and access keys, modify low-level silicon features, access unencrypted configuration bitstream or permanently damage the device."

So far, most countermeasure efforts for hardware trojans have been targeting application-specific integrated circuits (ASICs) and general-purpose microprocessors. In general, there have been two types of design-time countermeasures proved effective to detect custom hardware trojans: logic verification and side-channel analysis [4], [5]. In addition to suffering from high sensitivity to device and environment variations and low coverage of small/rare trojans, most of them rely on golden or reference models, which do not exist in the case of third-party IPs (3PIPs). FPGA trust and trojan insertion problems are inherently different because the FPGA design process is almost completely separate from the manufacturing flow, disallowing tampering during manufacturing and testing of the FPGA device [6].

Given the reconfigurability of FPGAs at both design time and run time, design-time detection methods alone appear insufficient. Even if all the malicious trojans have been detected and removed at design time and the bitstream has been protected by the encryption schemes provided by the FPGA vendors, it is still possible for an adversary to use side-channel attacks to recover the keys, and then, insert malicious logic into the bitstream [7]. There have been reported cases where the encryption schemes of both Xilinx and Altera FPGAs were broken. For instance, the full 128-bit Advanced Encryption Standard (AES) key of a Stratix II can be recovered in less than 3 h [7]. On the other hand, for mission-critical applications, recovery from abnormal state caused by activated Trojans is as important as detection because in many cases, the infected FPGA is expected to continue functioning correctly until it is replaced. Therefore, run-time trojan detection and recovery must be deployed to continuously monitor the FPGA's behavior and stop/remove trojans.

With the quick increase in chip density and deep scaling of feature size, current billion-transistor chip designs introduce more challenges to manufacturing fault-free chips. It is expected that, as the number of cores on a chip rapidly increases, packet switching wormhole-routing NoCs will replace current buses in the near future and become the interconnection network for future SoCs. NoCs, as any other silicon-based designs, are susceptible to faults. Such faults are categorized into two types: transient and permanent. Transient fault happens while the system is functioning, like electromagnetic interference as an example, while permanent fault, which is also the focus in this paper, is usually caused by either manufacturing defects or the accelerated aging effects of chips.

Since chips are becoming more and more susceptible to defects, stochastic and deterministic fault-tolerant routing algorithms have been proposed to provide reliable on-chip communication to NoC-based SoCs. A stochastic algorithm utilizes the idea of data redundancy and replicates packets in the network. Sending these packets over different routes in the network provides some sort of reliability. The major drawback of such algorithms is that when a relatively large amount of faults is found in a network, huge performance degradation is observed. In addition, such algorithms provide very high latency in moderately busy networks due to the idea of replication of packets. Deterministic algorithms use predefined rules to route around faulty blocks in a network. Results of such algorithms show a major precedence in terms of latency over the aforementioned stochastic algorithms [8].

In this paper, we introduce our integrated run-time solution for both security and fault-tolerance of FPGA-based SoCs through digital signatures, live monitoring, adaptive routing, and partial reconfiguration supported by an in-house-developed network on chip (NoC), X-Network [9]. Section II describes related research on security and fault tolerance of NoC-connected SoCs and the differences between our approach with those discussed. Section III details our design for IP security and trust. Section IV presents our adaptive fault-tolerant routing algorithm. Section V presents our run-time partial reconfiguration design for fault tolerance. Section VI presents experimental results of the integrated system. Conclusions are drawn in Section VII.

II. Related Work

Our research aims to provide an integrated run-time and architecture-level solution to both fault-tolerance and security of NoC-connected SoCs. Traditionally, these two aspects have been tackled separately. Therefore, we discuss the closest related work in the following two separate subsections.

A. *Security of SoCs*

With the mounting concerns about the devastating attacks on SoCs, increasing efforts are drawn to research against attacks of hardware trojans from various levels of design. Most countermeasures can be classified into design-time detection methods of trojans in IPs and design methodology for security [4]. Although such approaches have been proven effective for many types of known trojans, it is extremely difficult to guarantee a high coverage. Some trojans are not activated until a run-time physical condition is met. Run-time approaches are emerging to prevent and recover from hardware trojan attacks. We focus on run-time and architecture/system-level solutions in this study. Specifically for NoC-connected SoCs, most of these countermeasures modify the network interfaces in the processing elements (PEs) [10]–[15]. Specific security modules are added to instantaneously monitor the operation of such PEs. The main drawback of such dynamic and instantaneous checking is that they introduce significant degradation in performance if security breaches need to be detected preattack. To improve the performance of such mechanisms, the monitoring should be done in parallel, and therefore, the security breach will be detected after the attack would have started, which is not acceptable. In addition, the integration of such modules in each and every PE incurs a high area consumption.

We tackle the problem from the perspective of public key cryptography. In this paper, we use the Rivest-Shamir-Adleman (RSA) encryption/decryption algorithm as a form of digital signature for the IP cores present in the SoC. For this sake, it was vital to survey some of the best implementations of the RSA algorithm on FPGAs. Since RSA is a very resource- and time-consuming algorithm, several research groups have focused on its efficient implementation in hardware [16]–[22]. In [16], Song *et al.* presented an efficient and scalable hardware implementation of the RSA algorithm based on the Montgomery multiplication and a 1024-bit encryption/decryption runs in 36.37 ms at a maximum frequency of 447.027 MHz. Later the latency of the design was reduced to 11.263 ms at a maximum frequency of 410.678 MHz by using the Chinese remainder theorem [17].

In [18], Chow *et al.* propose a hardware implementation of the RSA algorithm based on the Karatsuba multiplication. This system reduces the efficiency class of the regular modular multiplication from $O(N^2)$ to $O(N^{\log 3/\log 2})$. It is scalable and has low lookup-table (LUT)-delay product and multiplier-delay product. Nakano *et al.* presented efficient hardware algorithms for modulo exponentiation used in RSA on FPGAs in [20]. Their main contributions are to present hardware algorithms based on the 64-K number system to complete the Montgomery modulo multiplication.

B. Fault-Tolerant NoC Designs

Driven by the critical importance of the communication network to the entire SoC, a significant research has been devoted to enhancing dependability of NoCs. Redundancy is a fundamental mechanism in most approaches for fault-tolerant NoCs to mitigate router and link failures and their impact on system's performance and reliability [8]. Some of the early and traditional work done on fault tolerance for NoCs was through the use of network flooding algorithms [23], [24] and stochastic algorithms [25], [26]. Those methods were used to overcome permanent and transient faults. The major problem introduced by such mechanisms is the introduction of high latency in the network due to the congestion created by sending packets more than once. To improve these drawbacks, researchers worked on improving partially and fully adaptive algorithms to accommodate to faults in the networks.

In [27], Wu proposed a deadlock-free and fault-tolerant deterministic routing algorithm based on the dimension order routing and odd–even turn model. Only faults happening on nodes are considered, and some nodes should never be faulty for the algorithm to work properly. An adaptive fault-tolerant deadlock-free routing algorithm for torus networks is presented in [28]. The network is divided into two virtual networks where packets are routed using two different routing algorithms, which require two sets of virtual channels in each physical channel, and hence, consumes a lot of extra area resources. The fault-tolerant routing scheme proposed in [29] is proven to be deadlock and livelock free; however, simulations show that good efficiency of this scheme is maintained up to only when 12% of network links are faulty. DeOrio *et al.* presented a fault-tolerant router architecture [30] that leverages reconfigurable architectural components to work around errors and reroutes packets around faulty nodes or links. However, the routing algorithm depends on storing data in routing tables to keep track of network status. This incurs high area utilization.

Other solutions that address permanent faults in NoCs make use of reconfiguration of the network topology or the router modules in the network [31], [32]. Kia *et al.* partition their network into segments of links and use spare wires to address permanent faults on specific bits of the links [31]. The major drawback of this approach is that in order to achieve adequate reliability in terms of fault tolerance, the area overhead would be large and ultimately increase power consumption too. Results show that the area overhead between segmented and nonsegmented links can be between 3.71% and 107.4% larger. Our design, however, does not introduce any area overhead, and thus, no increase in power consumption either. In [32], Stensgaard *et al.* proposed a design to reconfigure topology switches that are inserted at a layer between routers and links. This allows them to map different applications to different NoC topologies and obtain better performance for each application. Their design, however, imposes area overhead (10%) and a slight increase in power consumption (3%). In addition, the reconfiguration process happens when mapping the design and not at runtime.

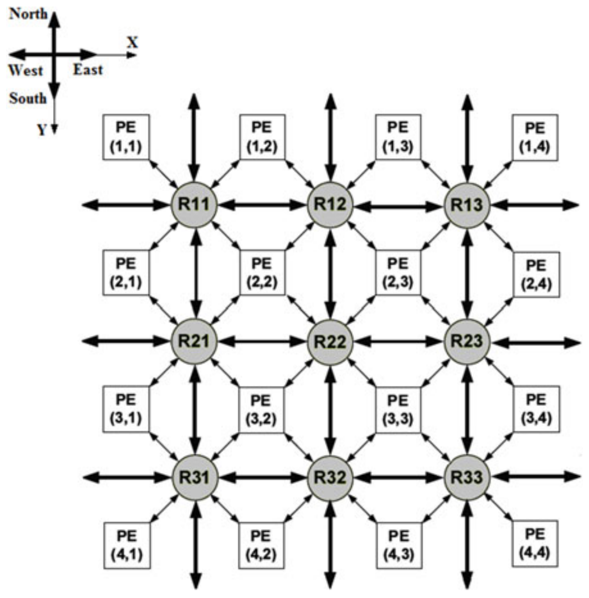

Fig. 1. 4 × 4 2-D Torus X-Network.

III. Our Design for IP Security and Trust

Our target network is the 2-D torus X-Network [9] shown in Fig. 1. We approach the idea of confidentiality of IPs by using public key cryptography and adapting it to NoCs implemented on FPGAs to prove the authenticity of the IP at configuration time. This reduces the performance overhead for checking the confidentiality of IPs at run time. In addition, we integrate security modules into our routers instead of integrating security features into the network interface of each PE.

Our proposed design covers all three properties of the security CIA (confidentiality, integrity, authentication) triad. The confidentiality and authenticity checks of the IP cores in the network are performed only during the configuration time of the FPGA or after partial reconfiguration. The IP vendor is asked to provide a public key generated by the asymmetrical RSA cryptographic algorithm. He will perform an offline encryption through his securely hidden private key for each of his IP cores. The public key is then used in our X-Network to authenticate the IP. When a specific IP is determined to be insecure, the network replies back with a countermeasure of partially reconfiguring its link with the network and disconnecting it.

The integrity of the IPs in the network is managed in the proposed design by instantaneously monitoring packets that run through the routers instead of IPs. This gives the advantage of reducing the area resource consumption since our network is constructed of 9 routers to serve 16 PEs. Therefore, we only need eight security modules integrated into 8 of our 9 routers to

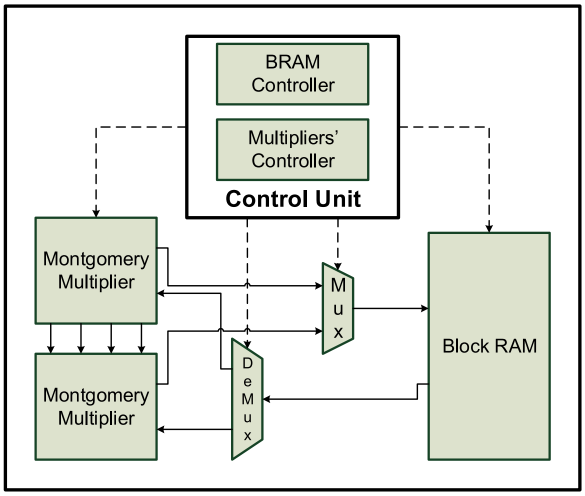

Fig. 2. Modular exponentiation implemented by two Montgomery multipliers and a BRAM.

cover the integrity of all 16 PEs (please refer to Section III-B and Fig. 5 for further details).

A. RSA Decryption for Confidentiality and Authenticity of the IP Cores

RSA operations are very computationally intensive and their hardware implementations are very resource expensive. High radix Montgomery multiplication can boost the performance significantly but with a high price of considerable resource increase. Integrating RSA engine into processors has been quite challenging because of limited available resources. Since our work targets Xilinx Virtex FPGAs, when implementing our RSA decryption engine, we adopt the radix 2^{17} Montgomery multiplication architecture designed by Song *et al.* [16], [17]. This architecture uses one DSP block and one block RAM (BRAM) to implement an RSA algorithm and makes the most efficient use of the embedded multipliers of current FGPA devices. Therefore, our system can be implemented on any state-of-the-art FPGA. In order to increase the speed of the decryption process, we adopt a multiplexed architecture [22] and modify both architectures to our needs. Integrating these two architectures together creates our unique implementation, which has the advantage of low area consumption and high performance.

Our RSA decryption engine is instantiated in a security module adjacent to our NoC. Fig. 2 shows the block diagram of our proposed architecture. Two modular multipliers are instantiated to work in parallel. They both share the same BRAM through a series of multiplexers (MUXes) and demultiplexers (DeMUXes). The control of the MUXes and DeMUXes comes from the control unit. The control unit is also responsible for providing the right address to the BRAM at the right time and managing the processes run in the Montgomery multiplier modules. Several iterations are performed by the multipliers to compute the modular exponentiation required by the RSA algorithm. When the result is computed, the control unit sends out a signal indicating the end of the decryption and the IP's signature is verified.

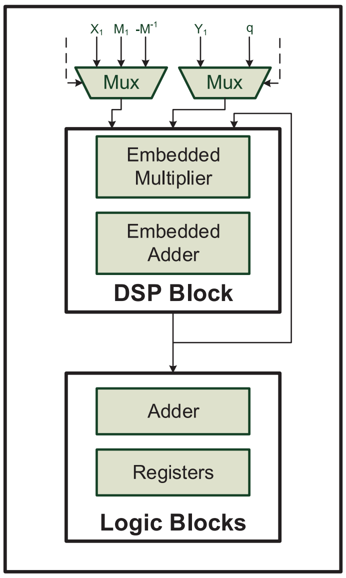

Fig. 3. Implementation of a Montgomery modular multiplier using one DSP block.

Fig. 3 shows the architecture of one modular multiplier modified to work as a radix 2^{17}. A single DSP block is used in each Montgomery multiplier. In addition, an adder and registers are instantiated using logic blocks to complement the work of the multipliers and generate the results.

As mentioned earlier, this novel architecture in NoCs provides authenticity of IP cores during the initialization stage of the network. Therefore, the signature decryption process and verification, happens before any transfer of packets to and from the newly added IP. The same thing happens in the case of partial reconfiguration. If a new IP is to be added to the network, the security module receives the signature and verifies it before completely attaching the IP to the network.

In the case where a signature shows a nonauthentic IP, the security module orders a Microblaze processor (an authentic IP in the network closest to the malicious IP) to run a program and perform partial reconfiguration to disconnect the links of that IP to the network. Fig. 4 shows an example where PE (1,1) turns out to be a malicious IP and partial reconfiguration is performed by PE (1,2) to disconnect its links to router (1,1).

Specifically, PE (1,2) uses the AXI Interconnect Block to reconfigure the reconfigurable partition. This happens through the two modules AXI_HWICAP and AXI_IPC. The connected reconfigurable module (Connected_RM) is exchanged with the disconnected reconfigurable module (Disconnected_RM) to ensure the disconnection of PE (1,2) from the network. It is to be noted that any packets intended to be transferred to the malicious PE (1,1) will be dropped by router (1,1).

B. Live Monitoring of the IP Cores in the X-Network

To provide integrity of packets coming in and out of the network, we implement security monitors at the level of the wormhole routers. These monitors are able to provide fault tolerance against transient faults too. The monitor in each router works

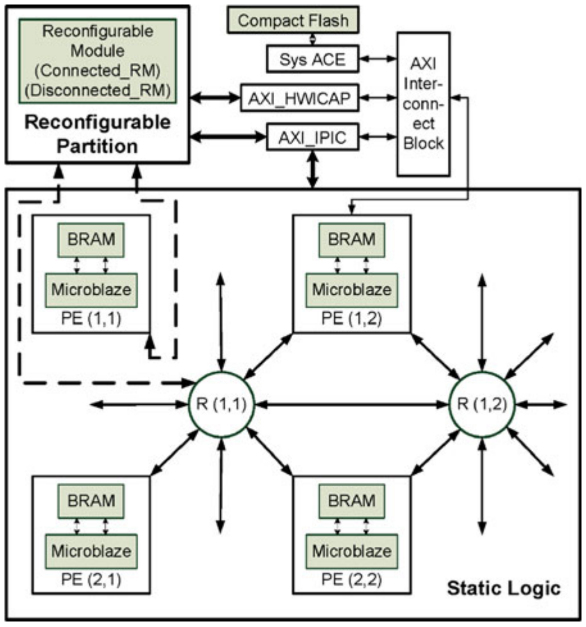

Fig. 4. Partial reconfiguration to disconnect a malicious IP.

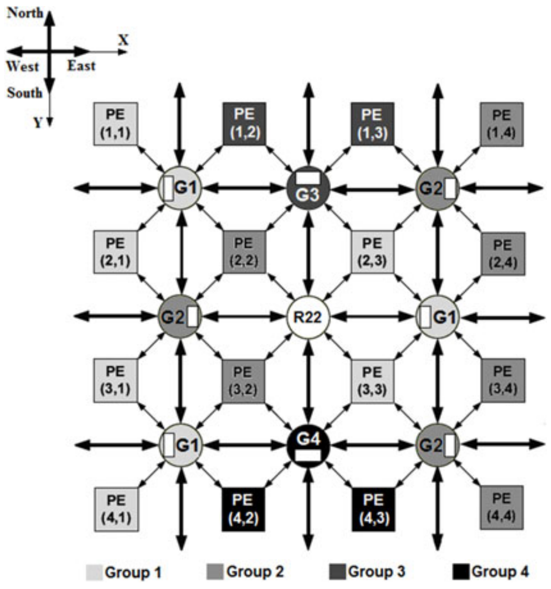

Fig. 5. Security monitors implemented in four different groups in the X-Network.

serially on securing two IPs that are connected to the router. The routers are divided into four groups as shown in Fig. 5. The first group consists of three routers that secure the two IPs found to their left. The second group consists of another three routers that secure the IPs to their right. The last two groups secure the two IPs located to their north and south region, respectively. As Fig. 5 shows, all the IPs in our 4×4 2-D Torus X-Network are now monitored using the proposed scheme. The only router that does not implement a security monitor in its modules is the center router and this provides a performance advantage since it is usually the most congested.

Our security monitors check the source and destination address of each packet and prove that the sending IP has the right to

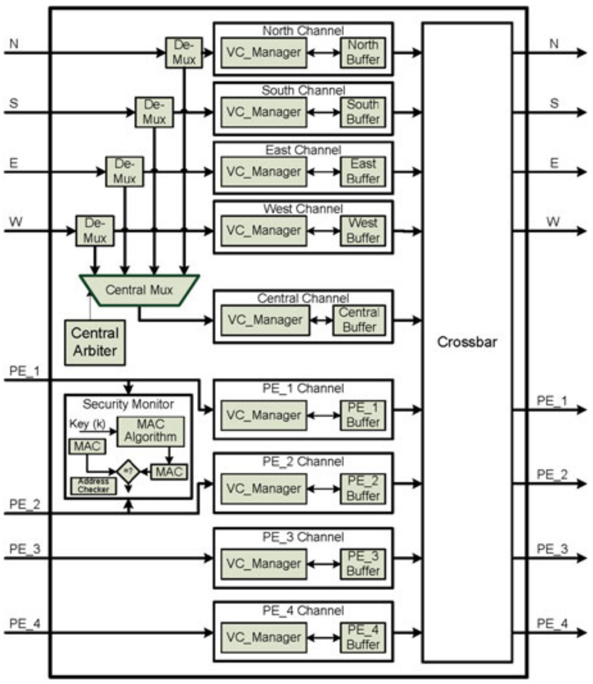

Fig. 6. Security monitor shared between two PEs inside a router of group 3.

communicate with the receiving IP. This is done by monitoring each source and destination pair and checking it with a database containing valid source/destination pairs stored in an attached BRAM. The monitors also check the data parts of each flit to verify that the sent data are not modified (achieves integrity and fault tolerance). This is performed by verifying the message authentication code (MAC) appended to the packet. Fig. 6 shows a detailed block diagram of the internal structure of a router. The north, south, east, and west channels are responsible for momentarily storing and routing flits coming from the north, south, east, and west links, respectively. The central channel is shared between all the links and used when any of the links is congested. Similarly, PE1, PE2, PE3, and PE4 channels are used to store and route flits coming from their corresponding PE links. As we mentioned earlier, routers are divided into four groups according to the position of the PEs they monitor. In the diagram of Fig. 6, the monitor is implemented in a router of group 3. This means that it monitors PE1 and PE2, which are the PEs connected to the north of the router. When every packet enters one of the two channels, it continues its normal path toward arbitration and routing. In the meantime, the packet is processed inside the security monitor and the data part is checked for its integrity using the MAC algorithm and by comparing the generated MAC with the one appended with the message by the IP. It is noted that each of these monitors work serially on accepting packets coming from two IPs. The process happens during the arbitration and routing inside the router; therefore, imposing a minimal performance degradation.

IV. Fault-Tolerant Routing Algorithm

The proposed routing algorithm utilizes the conventional deterministic *XY* routing algorithm and adds the idea of adaptability to it in order to face network faults. The turn model-based routing algorithm is utilized to route around any faulty block or link in the network. The route, through which the packets take to avoid such faults, depends on the original deterministically selected direction for routing and on the physical channel through which the packet arrives into the router. The turn model routing algorithm can be of many variations. Glass and Ni present several routing algorithms that depend on the turn model in their work in [33]. In this paper, we use the west-first turn model to enhance the adaptability of the deterministic *XY* routing algorithm. The west-first routing algorithm states that packets are routed in the west direction first, if needed, before any other directions. When west routing is no longer needed, the packets can be routed dynamically in any of the north, south, or east directions. This type of turn model basically inhibits two turns out of the eight possible turns a packet can make. That is enough to break all cycles in the network.

In order to apply fault tolerance to the previously designed router, some major modules had to be changed. Mainly, the direction selector is the module responsible for routing the packets in each channel. Originally, the direction selector is shared among all the channels when it is implementing the *XY* routing algorithm. Now, however, the direction selector has a slightly different logic in each channel depending on the input channel in which it is instantiated.

The direction selector works in parallel with the virtual channel allocator. When the state machine of any of the virtual channels requests the direction selector, it calculates the route through which the packet is to be released. When the calculation finishes, a request is sent to the needed virtual channel allocator. The routing calculation depends on the current address of the router and the destination address of the PE. It is noted here that the destination address is held in the head flit of each packet. The routing happens in accordance to the *XY* routing algorithm where if the destination's *X*-coordinate is lower than the current router's *X*-coordinate, the packet is routed west. On the contrary, if the destination has a higher *X*-coordinate compared to the router, the packet is routed east. When the packet reaches the router that has the same *X*-coordinate or a coordinate that is higher by one (X-Network), it is routed in the *Y*-direction. For example, suppose the destination of a packet is PE (1,3). If the packet arrives to router R(3,2), or R(3,3) (see Fig. 1), it no longer needs to travel along the *X*-direction and can thus be routed in the *Y*-direction because it will eventually reach a router that is directly connected to the destination PE. The same process happens to determine whether the packet is to be routed north or south. However, this time the *Y*-coordinates are to be compared. Since our network is a 2-D Torus X-Network, border routers are treated a bit differently and packets are sometimes sent in the opposite direction of what the *XY* routing algorithm proposes. For example, when a packet that has an *X*-coordinate destination of 1 reaches router $(2, 3)$, it is routed to the east instead of west although the destination's *X*-coordinate is lower than the current router's *X*-coordinate. This is due to the wraparound connection between router $(2, 1)$ and router $(2, 3)$.

In order to achieve fault tolerance and better performance in terms of latency, the direction selector in each physical channel of a router checks the status of neighboring routers in order to determine if an upstream router is busy or faulty. If that is the case, the direction selector tries to give the packet a different route in order to reach the destination.

TABLE I
DIRECTION OF ROUTING AFTER APPLYING THE ENHANCED-*XY* ROUTING ALGORITHM

Input Port	Blocked Original Direction of Routing	New Direction of Routing			
		North	South	East	West
East	North	–	X	–	–
	South	X	–	–	–
	West	X[1]	X[1]	–	–
West	North	–	X	–	–
	South	X	–	–	–
	East	X[1]	X[1]	–	–
North	South	–	–	X[2]	X[2]
	East	–	X	–	–
	West	–	X	–	–
South	North	–	–	X[2]	X[2]
	East	X	–	–	–
	West	X	–	–	–

[1]Decision depends on the Y coordinates of the source and destination.
[2]Decision depends on the X coordinates of the source and destination.

Table I shows the new functionality of the direction selectors in each input channel. The direction selectors instantiated in the east and west input channels have the same functionality. When a packet is directed to the north and the upstream router is faulty or the link transmitting the packets to that upstream router is faulty, the packet is directed south. When a packet is directed to the south and the upstream router is faulty or the link transmitting the packets to that upstream router is faulty, the packet is directed north. When a packet is in the east channel and going to a blocked west channel, the packet is routed south if the destination's *Y*-coordinate is higher than the current router's *Y*-coordinate; else it is routed north. The same thing happens when the packet is in the west channel and needs to be routed to the east channel. Direction selectors instantiated in the north and south input channels have different implementations. The direction selector used in the north channel sends packets that are originally destined to a blocked south channel in an east or west direction depending on the current router's *X*-coordinate and the destination's *X*-coordinate. Moreover, when a packet in the north channel is destined toward a blocked east or west channel, the packet is routed south. The last variation of the direction selector is used in the south channel. When a packet in the south channel is destined to a blocked north channel, it is routed to the east or west depending on the current router's position and the destination address. Similarly, if the packet is destined to a blocked east or west channel, it is routed in the north direction to prevent the faulty links.

V. Run-Time Partial Reconfiguration for Fault Tolerance

One of the major advantages of FPGAs over ASICs is their reconfigurability, partially or entirely, at both design and run times. Partial reconfiguration is the process of configuring one (reconfigurable) part of the FPGA while the main (static) part is running and functional. Usually, reconfiguration of FPGAs requires the whole system to be reset in order for some external hardware components to load the FPGA with a new design. Partial reconfiguration, however, allows for some reconfigurable modules to be reloaded with new designs while the rest of the static modules are operating. Partial reconfiguration uses a partial configuration file (a partial bit file) to modify reconfigurable regions in the FPGA after loading a full bit file to configure the FPGA. Some of the major advantages that partial reconfiguration gives, by providing the ability to time multiplex hardware dynamically on a single FPGA, are as follows:

1) reducing the size of the FPGA device required to implement a given function, with consequent reductions in cost and power consumption;
2) providing flexibility in the choices of algorithms or protocols available to an application;
3) enabling new techniques in design security;
4) improving FPGA fault tolerance;
5) accelerating configurable computing.

When building a design that holds reconfigurable modules, special care should be taken. Each reconfigurable module for a reconfigurable partition must have the same set of module ports. In addition, all external port declarations should be made at the top level. Moreover, automatic input/output block (IOB) insertion should be enabled for the top-level synthesis and disabled for module level synthesis.

The major goal behind partial reconfiguration is to make efficient use of FPGA hardware resources by modifying a reconfigurable part in the design to implement different functions at different times. Reconfiguring FPGA hardware resources can be compared to how a microprocessor switches between different tasks. Due to the ability of reconfiguration in FPGA devices, and due to its ability of switching tasks in hardware, the benefit of both flexibility of a software implementation and the performance of a hardware implementation is obtained. Our design takes advantage of the power of this technology.

The enhanced-*XY* routing algorithm succeeds in providing great fault tolerance in the X-Network as long as PEs manage to stay connected to the network as results show in Section IV. However, if a link or a router fault happens to one of the routers connecting corner PEs, this PE would be eliminated from the network (see Fig. 7). Therefore, no packets destined to that PE would be able to reach it. To solve this problem, we present a reconfigurable architecture that takes advantage of partial reconfiguration. For example, as Fig. 7 shows, our design manages to reconnect PE $(1,1)$ to the network through the fourth connection of router $(1,2)$ after router $(1,1)$ becomes faulty. To do so, PE $(2,2)$ has to be disconnected from this router and remain connected to the network through the two routers $(2,1)$ and $(2,2)$. Although partial reconfiguration is widely known in introducing latency overhead, in our case, the process does not interfere with the flow of data in the network. All the work is done on the links that are already down due to a router failure. The only minor drawback that might slightly increase the average network latency in rare scenarios when congestion is very

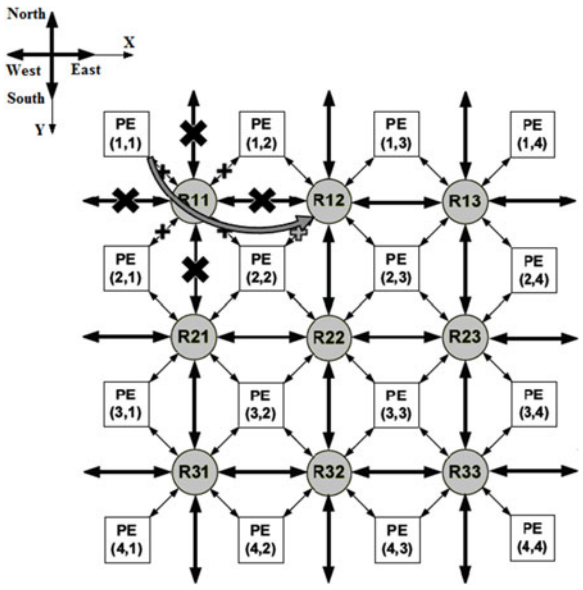

Fig. 7. Applying the reconfiguration process: when router $(1,1)$ is faulty, PE $(2,2)$ is disconnected from router $(1,2)$ and PE $(1,1)$ is connected in its place.

high, is the disconnection of only one link between PE $(2,2)$ and the network. This is of minor importance since the center PE is still connected to the network through two other routers.

Our partial reconfiguration design uses the Xilinx Platform Studio (XPS), IP CORE Generator software development kit (SDK), and the PlanAhead design tool. We use XPS to create a processor hardware system that includes a lower-level module defining one reconfigurable partition (RP) holding two reconfigurable modules (RMs) and a static logic (SL) partition. The two RMs represent the two different states of the system. The IP CORE Generator is used to code and synthesize these two modules to be connected and configured in the design. We also use an SDK to create a software application that enables the dynamic partial reconfiguration and computes the time taken during the switch between the two states. We use PlanAhead to floorplan the design including defining a reconfigurable partition for the reconfigurable region, and to create multiple configurations and run the partial reconfiguration implementation flow to generate full and partial bitstreams. Our design is finally tested on the Xilinx Virtex-6 FPGA to verify it in hardware using a compact flash (CF) memory card to configure the FPGA device initially, and then, partially reconfigure the device using the AXI HW-ICAP peripheral by loading the partial bitstream files stored on the CF under the software control. Fig. 8 shows the top level design of the processor system and the connected peripherals. The figure also shows part of the static logic and the reconfigurable partition related to the case when router $(1,1)$ gets faulty.

The processor used is a Xilinx Microblaze processor that might be running individually or on a PE (in our case). The C code (see flowchart in Fig. 9) interfaces with the hardware using a user IP peripheral and shared registers to monitor corner routers' credits. The software checks these credits every specific amount of time. If the credits show that the router has all its buffers full for a relatively long time, it assumes that the router is faulty and starts the reconfiguration process. The first

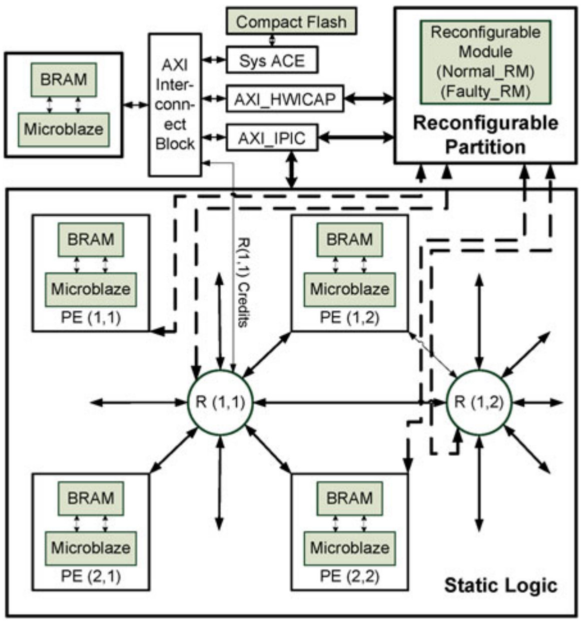

Fig. 8. Detailed view of the connection between the Xilinx processor and its peripherals to perform dynamic partial reconfiguration of the reconfigurable partition (RP).

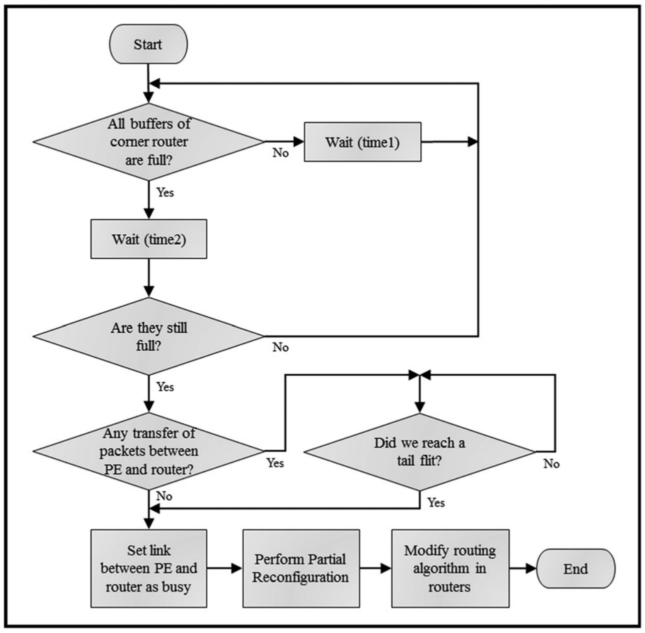
Start
All buffers of corner router are full?
No
Wait (time1)
Yes
Wait (time2)
Are they still full?
No
Yes
Any transfer of packets between PE and router?
Yes
Did we reach a tail flit?
No
No
Yes
Set link between PE and router as busy
Perform Partial Reconfiguration
Modify routing algorithm in routers
End

Fig. 9. Logic behind the main controller for partial reconfiguration.

step is to monitor any transfer of packets between the new router (router $(1, 2)$ in our case), to which the corner PE (PE $(1, 1)$) is going to be connected, and the old PE (PE $(2, 2)$), which is going to be disconnected. If a packet is being transmitted on this link, the process is suspended until the tail flit of the packet is reached. At this point, the controller sets the link between the PE and the router as busy to prevent any packets from being transmitted. The second step is to perform the reconfiguration process by swapping the intermediate reconfigurable modules to reconnect the new PE to the new router. The swapping happens through the AXI_IPIC interface by the means of the CF, Sys ACE, AXI_HWICAP, and the AXI Interconnect Block

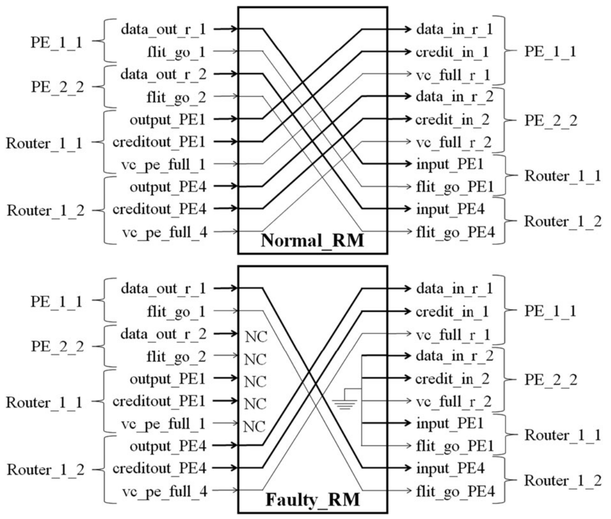

Fig. 10. Reconfigurable modules that are used in the case of a faulty router $(1, 1)$.

(see Fig. 8). The reconfigurable modules shown in Fig. 10 are stored on the CF and are loaded through the AXI modules to the reconfigurable partition as needed. In this case, the reconfiguration process basically disconnects the data inputs and outputs and the credits of the router from the old PE and connects them to the new PE (see Fig. 10). Finally, after the reconfiguration process is successful, the controller sends a signal to all the routers in the network to identify the new connection of the PE. This will modify the behavior of the routing algorithm to reroute the packets to the correct new destination of the newly connected PE. It also changes the method of sinking packets into directly connected PEs for the affected routers.

The enhanced-*XY* algorithm alone was tested against different faults in the network under uniform random traffic. Two types of faults were simulated to test the efficiency of the design: faults happening to links that connect different routers and faults happening to routers themselves. In order to simulate a faulty link or router, the credits coming from an upstream router into a downstream router were used. For example, if the link that sends packets out of an east channel of a router to a west channel of another router is to be made faulty, the input credits coming into the east channel of the downstream router are set to zeros indicating that the upstream router is always busy. Similarly, if a router is to be made faulty, it is forced to send credits to all neighboring routers and PEs saying that it is full at all times. Different scenarios and fault percentages were simulated. Section VI shows all the simulation parameters and the extracted results.

VI. Experimental Results and Analysis

All of the following experiments were conducted on the Xilinx ML605 FPGA development board, which features a Virtex-6 FPGA.

A. *Secure Design*

The following section presents the experimental results of the wormhole router after implementing the security features

TABLE II
RESOURCE CONSUMPTION OF THE MONITORS IN A SINGLE WORMHOLE ROUTER

Resources	Without Monitors	With Monitors	Overhead (%)
Slice registers	8 703	8 846	1.64
LUTs	19 975	21 689	8.58
36-Kb BRAMs	9	10	11.11

TABLE III
RESOURCE CONSUMPTION OF THE SECURE AND INSECURE NETWORKS

Resources	Normal Design	Secure Design	Overhead (%)
Slice registers	78 318	83 186	6.22
LUTs	178 060	199 069	11.8
36-Kb BRAMs	81	95	17.28
DSP48E1 blocks	0	19	NA

of both, confidentiality of IPs and integrity of packets. It is to be noted that this design also includes the same logic that achieves fault tolerance. Therefore, the partial reconfiguration architecture is used for both, fault tolerance and malicious IP elimination (compare Figs. 4 and 8). This reduces the resources needed for both designs. Since security has a higher priority in the design, it is given the precedence in taking control of the software running the partial reconfiguration on the secure processor. For example, if the processor features a malicious IP in its region, the partial reconfiguration program runs to disconnect this IP ignoring any other requests to reconnect a defective IP back to the network.

Table II shows the difference in resource consumption between a normal fault-tolerant router and a router implementing a security monitor on its interfaces. The overhead percentages shown in Table II are calculated using the following formula:

$$\text{Overhead} = \frac{\text{With Monitors} - \text{Without Monitors}}{\text{Without Monitors}} \times 100.$$

Additional resource usage is basically due to the implementation of the MAC algorithm, which checks for packet integrity. Since our monitors are implemented on the router side and not on the IP side, the resource consumption of the whole network is reduced by half. Specifically in our 4×4 2-D Torus X-Network, we are only using eight monitors as compared to the use of 16 monitors in conventional network security designs.

As mentioned earlier, the RSA decryption engine is only used when a new IP is introduced to the system or after partial reconfiguration happens. Therefore, the use of one decryption module is enough to support the security of the whole design in a relatively reasonable time. Our simulation results show that the running time of each decryption process of a 1024-bit digital signature on our engine clocked at 100 MHz (maximum frequency 114 MHz) requires on average 5.85 ms. When compared to other research work [16], [17], [22], it is clear that our architecture provides a compromise between resource usage and speed of computation resulting in an architecture that satisfies our area and speed needs. For example, in [16], a 1024-bit modular exponentiator algorithm required the use of 1 BRAM and 1 DSP48E1 block but was able to perform the decryption in 36.37 ms running at a frequency of 447.027 MHz. While in [22], a 1024-bit modular exponentiator was able to decrypt a signature in 4.36 ms but used 64 DSP48 blocks and 512 bytes of BRAM.

Table III shows the resource consumption of the whole 4×4 2-D Torus X-Network after implementing all the modules responsible for fault tolerance and security. Specifically, the results take into account the resource usage of the fault-tolerant algorithm and the security monitors in each router, the RSA decryption engine, the reconfigurable blocks, and their connections. The overhead percentages shown in Table III are calculated using the following formula:

$$\text{Overhead} = \frac{\text{Secure Design} - \text{Normal Design}}{\text{Normal Design}} \times 100.$$

B. Fault-Tolerant Design

The following section presents the ModelSim simulation results of the wormhole router in VHDL after implementing the enhanced-*XY* routing algorithm to test the fault tolerance of the X-Network. It then shows the advantage of adding the reconfigurable architecture to the design in order to improve the fault tolerance specifically when corner routers get faulty. The type of traffic used in the network for the purpose of simulation is the uniform random traffic. Two types of faults were introduced with different varying percentages and positions in the network. The percentage of faults happening to links that connect routers was varied between 5, 10, and 20 percent. In addition, three different scenarios were simulated where a corner router, a border router, and a center router were made faulty. The evaluation of the network's performance in all the scenarios was done by comparing the average latencies and the saturation points of the network. To enable precise comparisons, the saturation point is extracted graphically (Figs. 11–14) by extending a line along the exponential increase in injection rate and reading out the value of the intersection point of the extended line with the *x*-axis.

Our 2-D Torus X-Network connects 3×3 routers and 4×4 PEs. Thus, there are 18 bidirectional or 36 unidirectional links between these routers (see Fig. 1). To simulate a network with 5% faults, two unidirectional links were chosen randomly from the network and made faulty. Similarly, four and eight unidirectional links were chosen to represent the 10 and 20% percent faults, respectively. As for routers, router $(1, 1)$ was chosen as the faulty corner router, router $(2, 2)$ was chosen as the faulty center router, and router $(2, 3)$ was chosen as the faulty border router.

Fig. 11 shows a graph that represents the variation of the average latency of a packet in the network with respect to injection rate at different link fault percentages. Results show that when 5% of the links in the network are faulty, performance is slightly degraded and the latency increases after 45% injection rate to reach a saturation point at 55%. This is so close to the one reached when no faults were present in the network. The curve

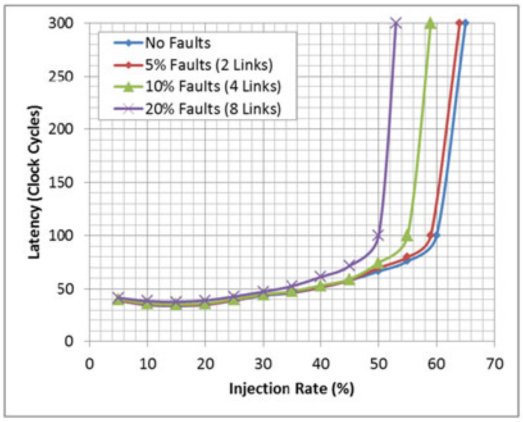

Fig. 11. Average latency of the network in the presence of link faults.

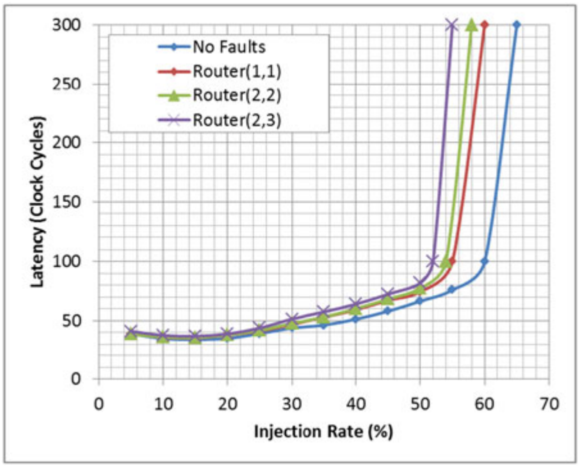

Fig. 12. Average latency of the network in the presence of router faults.

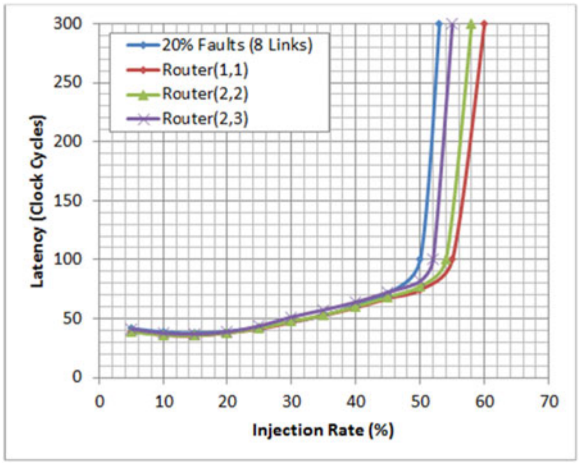

Fig. 13. Comparison between 20% link faults (eight links) and router faults.

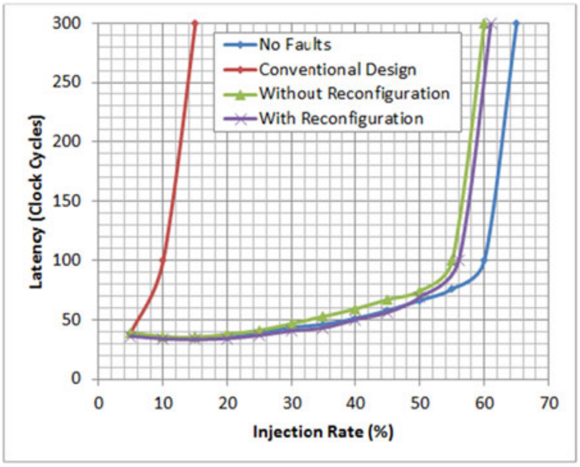

Fig. 14. Network reaction to different design scenarios when router $(1, 1)$ is faulty.

with the 10% faults shows higher performance degradation and a sooner saturation point (50%). This is expected since when the number of faults in the network increases, the packets need to be routed around these faults resulting in a higher delay to reach their destination. This would eventually result in a higher congestion rate in the network leading to earlier saturation points. The last curve shows the performance of the X-Network when the percentage of faults is 20. The average latency of packets in the network increases sooner in such scenario starting at 30% injection rate leading to an early saturation at 45% injection rate. Although 20% is considered a high percentage of faults to be found on links in an on-chip network, the enhanced routing algorithm combined with the original design of the X-Network manage to provide very good results as compared to other work such as in [29] and [30].

Fig. 12 shows the average latency curve of the network when faults are occurring at routers $(1, 1)$, $(2, 2)$, and $(2, 3)$. Comparing all three scenarios to the conventional router design, we can see that all of them feature a latency increase starting at 30% injection rate. However, the major difference is in the saturation point. Router $(1, 1)$, a corner router, has the highest saturation point reaching an injection rate of 52%. It is to be noted here that when simulating faults on router $(1, 1)$ in this case, the corner PE $(1, 1)$ is isolated since it only connects to the network through router $(1, 1)$. Therefore, to have a fair comparison, all other PEs were banned of sending packets to PE $(1, 1)$ since there is no way for the packets to reach the PE. This, however, leads to effectively the loss of one core in the SoC. Therefore, it is not favorable. The proposed reconfigurable architecture manages to solve this problem and reconnects the PE back to the network. Simulation results, analysis, and comparisons are presented later in this section. As for router $(2, 2)$, a center router, the graph shows that the saturation point was at 50% injection rate. This shows that the implemented enhanced-*XY* routing algorithm gives a better fault tolerance for center routers. This is also backed up by the nature of the X-Network which supports higher fault tolerance for routers connected to PEs in the center of the network. For example, the PEs connected to router $(2, 2)$ have three other routers to send and receive packets through. Although the malfunctioning of a center router has the worst performance in terms of latency since it is usually the busiest in the network, the enhanced-*XY* routing algorithm adapted to the X-Network manages to reroute and redistribute packets around the faulty router to provide a good overall performance while maintaining a high saturation point. The last simulated faulty router was the border router $(2, 3)$. This scenario shows the lowest saturation point (48%) and the highest latency. That is because two of the PEs served by this router (the border PEs), are only connected to one other router. For example, PE $(2,4)$ would be only connected to router $(1,3)$ when router $(2,3)$ is made faulty. Therefore, all packets heading to this PE need to be rerouted to pass through router $(1,3)$ to reach their destination. This increases congestion in the network, which raises the latency and ultimately leads to an earlier saturation point.

Fig. 13 shows a comparison between router and link faults for the same number of link faults. The results show that the network is more affected when such faults happen to links rather than routers. When any of these routers is made faulty, this means that the eight links connecting this router to other routers are made faulty. This is similar to the situation in Fig. 11 where 20% of the links in the network are made faulty. However, results are different. When eight faults affect eight different unidirectional links, the network saturates at an earlier stage with roughly 45% injection rate. When those eight links affect the same router (see Fig. 12), the network has a higher saturation point ranging between 48% and 52%. This is mainly due to the fact that when routers are made faulty, directly connected PEs know of their situation and stop sending packets toward them. However, when a link that connects two routers is made faulty, the PE never knows about the situation and continues to send packets at the same rate until the router gets congested leading to earlier saturation in the network.

Fig. 14 shows a comparison between different network reactions of different designs when router $(1,1)$ is faulty. By nature, when router $(1,1)$ goes faulty, PE $(1,1)$ gets disconnected from the network. This will directly lead to an early saturation in the network, even at low injection rates. In our case, the figure shows that the network saturates early at 8%. In our first design and before adding reconfigurability, where PE $(1,1)$ was still excluded from the network, the problem was solved. However, it introduces a software performance degradation due to the loss of one PE. The packets still reach all other PEs with a very good latency. After implementing the partial reconfiguration design, PE $(1,1)$ is reconnected to the network and the packets can reach it normally. The network saturation in this design happens at around 55% injection rate.

Another interesting metric that comes to mind when dealing with fault tolerance in NoCs is the percentage of successfully delivered packets under different link fault percentages. Fig. 15 shows that before reaching the 50% injection rate (the point where saturation happens), all packets were received during all simulation scenarios. This again proves the efficiency of our fault-tolerant design. At 50% injection rate, the design with 20% link faults shows saturation and around 2% of the packets do not reach their destination. Similarly, at 55% injection rate, packets in the designs with 10% and 20% link faults start missing their destinations. Starting at 60% injection rate, all designs including the one with no faults feature at least 3% loss in packet count. Yet, the percentages among all designs is still so close. The biggest difference in the percentage of received packets happens evidently at the 100% injection rate. The design with no faults shows a loss of approximately 4% in packet count while the design with 20% link faults shows the loss of around 6% of packets. When comparing our results to similar work [24], one can see that our design clearly has better

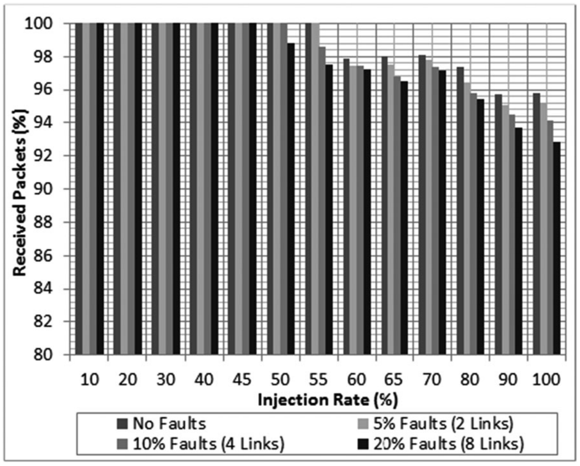

Fig. 15. Percentage of successfully received packets under different link fault percentages.

TABLE IV
RESOURCE CONSUMPTION BETWEEN THE NORMAL AND FAULT-TOLERANT DESIGN

Resources per Wormhole Router	Normal Design	Fault-Tolerant Design	Overhead (%)
Slice registers	8 702	8 703	0.011
LUTs	19 712	19 975	1.334
36-Kb BRAMs	9	9	0

reliability and fault tolerance. Even at very high injection rates and at up to 20% link faults, the percentage of successfully received packets never drops below a threshold of 92%. It is to be noted that an acceptable percentage of received packets is typically application dependent. For example, for a Voice over IP traffic application, if 98% of the sent packets are received, the quality of the conversation will not be affected. However, if only 90% of the packets are received the quality of the call will be affected significantly [34]. Therefore, for such type of applications, our fault-tolerant architecture is able to successfully meet the application requirements for injection rates of up to 55%.

Our design was implemented on a Xilinx Virtex-6 FPGA. The area comparison between the conventional router and the fault-tolerant router showed negligible difference (see Table IV). The router's logic changes minorly since the implementation of the fault-tolerant algorithm changes the logic only and does not add to it. In addition, the reconfiguration process is carried out by an external controller run on the software of the processor. Therefore, it in turn does not incur any area overhead. Experiments showed that the reconfiguration process for switching between the two reconfigurable modules when a router gets faulty takes around 146 884 clock cycles which is around 1.47 μs when the processor is operated using a 100-MHz clock.

VII. CONCLUSION

Technology scaling into the deep submicron range has enabled billions of transistor integrations. More and more concerns are arising about the security and dependability of such chips. By taking advantage of the flexibility offered by the in-house developed X-Network, we have developed an integrated run-time and architectural-level solution to enhance security and fault tolerance of NoC-based SoCs. Such run-time approaches provide more assurance against diverse natures of attacks from hardware trojans. The reduced ratio of routers versus PEs in X-Network not only leads to better performance, but also offers much more flexibility to support fault tolerance and security features with little overhead. Our proposed design covers all three properties of the security CIA through digital signatures, live monitoring, and partial reconfiguration. A multiplexed Montgomery modular multiplication architecture is used to increase the speed of the RSA algorithm. Implementation results on the Virtex-6 FPGA board show that the network manages to transmit packets from sources to destinations even when the percentage of faulty links goes up to as much as 20%. In addition, the proposed algorithm shows that it is even immune to node (router) failures, whether the faulty node is a center, corner, or border router. Both the security and fault-tolerant features incur minimal resource overhead and performance degradation.

References

[1] S. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," *Proc. IEEE*, vol. 103, no. 3, pp. 318–331, Mar. 2015.

[2] M. Zhang, A. Raghunathan, and N. Jha, "Trustworthiness of medical devices and body area networks," *Proc. IEEE*, vol. 102, no. 8, pp. 1174–1188, Aug. 2014.

[3] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proc. Cryptographic Hardware Embedded Syst.*, 2012, pp. 23–40.

[4] S. Bhunia, M. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[5] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.

[6] S. Trimberger and J. Moore, "FPGA security: Motivations, features, and applications," *Proc. IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug. 2014.

[7] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, "Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: Facilitating black-box analysis using software reverse-engineering," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2013, pp. 91–100.

[8] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in networks-on-chip," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 8-1–8-38, Jul. 2013.

[9] X. Wang and L. Bandi, "X-network: An area-efficient and high-performance on-chip wormhole interconnect network," *J. Microprocess. Microsyst.*, vol. 37, no. 8, pp. 1208–1218, 2013.

[10] J. -P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "NOC-centric security of reconfigurable SoC," in *Proc. Int. Symp. Netw.-on-Chip*, May 2007, pp. 223–232.

[11] S. Lukovic and N. Christianos, "Enhancing network-on-chip components to support security of processing elements," in *Proc. Workshop Embedded Syst. Security*, 2010, pp. 12-1–12-9.

[12] L. Fiorin, C. Silvano, and M. Sami, "Security aspects in networks-on-chips: Overview and proposals for secure implementations," in *Proc. Euromicro Conf. Digit. Syst. Des. Archit., Methods Tools*, Aug 2007, pp. 539–542.

[13] C. H. Gebotys and Y. Zhang, "Security wrappers and power analysis for SoC technologies," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2003, pp. 162–167.

[14] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for NoCs," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2008, pp. 197–202.

[15] S. Evain and J.-P. Diguet, "From NoC security analysis to design solutions," in *Proc. IEEE Workshop Signal Process. Syst. Des. Implementation*, Nov. 2005, pp. 166–171.

[16] B. Song, K. Kawakami, K. Nakano, and Y. Ito, "An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA," in *Proc. Int. Conf. Netw. Comput.*, Nov. 2010, pp. 140–147.

[17] B. Song, Y. Ito, and K. Nakano, "CRT-based DSP decryption using Montgomery modular multiplication on the FPGA," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 532–541.

[18] G. C. T. Chow, K. Eguro, W. Luk, and P. Leong, "A Karatsuba-based Montgomery multiplier," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 434–437.

[19] G. Iana, P. Anghelescu, and G. Serban, "RSA encryption algorithm implemented on FPGA," in *Proc. Int. Conf. Appl. Electron.*, Sep. 2011, pp. 1–4.

[20] K. Nakano, K. Kawakami, and K. Shigemoto, "RSA encryption and decryption using the redundant number system on the FPGA," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–8.

[21] T. Alho, P. Hämäläinen, M. Hännikäinen, and T. D. Hämäläinen, "Compact modular exponentiation accelerator for modern FPGA devices," *Comput. Elect. Eng.*, vol. 33, no. 5–6, pp. 383–391, Sep. 2007.

[22] G. Perin, D. G. Mesquita, and J. a. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: Systolic versus multiplexed implementation," *Int. J. Reconfigurable Comput.*, vol. 2011, pp. 6–1–6–10, Jan. 2011.

[23] A. Sanusi and M. Bayoumi, "Smart-flooding: A novel scheme for fault-tolerant NoCs," in *Proc. IEEE Int. SOC Conf.*, Sep. 2009, pp. 259–262.

[24] M. Pirretti *et al.*, "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Feb. 2004, pp. 46–51.

[25] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant networks-on-chip," *VLSI Des.*, vol. 2007, no. 95348, 2007.

[26] W. Song, D. Edwards, J. Nunez-Yanez, and S. Dasgupta, "Adaptive stochastic routing in fault-tolerant on-chip networks," in *Proc. ACM/IEEE Int. Symp. Netw.-on-Chip*, May 2009, pp. 32–37.

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

[28] J. -D. Shih, "Adaptive fault-tolerant wormhole routing for torus networks," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 1998, pp. 558–565.

[29] A. Vitkovskiy, V. Soteriou, and C. Nicopoulos, "Dynamic fault-tolerant routing algorithm for networks-on-chip based on localized detouring paths," *IET Comput. Digit. Tech.*, vol. 7, no. 2, pp. 93–103, 2013.

[30] A. DeOrio *et al.*, "A reliable routing architecture and algorithm for NoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 31, no. 5, pp. 726–739, May 2012.

[31] H. S. Kia and C. Ababei, "Improving fault tolerance of network-on-chip links via minimal redundancy and reconfiguration," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2011, pp. 363–368.

[32] M. Stensgaard and J. Sparso, "ReNoC: A network-on-chip architecture with reconfigurable topology," in *Proc. ACM/IEEE Int. Symp. Netw.-on-Chip*, Apr. 2008, pp. 55–64.

[33] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *J. ACM*, vol. 41, no. 5, pp. 874–902, Sep. 1994.

[34] K. C. Mansfield and J. L. Antonakos, *Computer Networking for LANs to WANs: Hardware, Software and Security*, 1st ed. Clifton Park, NY, USA: Delmar Learning, 2009.

Taimour Wehbe (S'10) received the Bachelor's degree in computer and communications engineering from the American University of Science and Technology (AUST), Beirut, Lebanon, in June 2011 and the Master's degree in electrical engineering from Villanova University, Villanova, PA, USA, in December 2013. He is currently working toward the Ph.D. degree majoring in electrical and computer engineering at the Georgia Institute of Technology, Atlanta, GA, USA.

His research interests include hardware security and trust, embedded systems security, VLSI design, reconfigurable and fault-tolerant computing, and systems-on-chip.

Xiaofang (Maggie) Wang (S'01–M'06) received the B.S. degree in microelectronics from Nankai University, Tianjin, China, the M.S. degree in electrical engineering from the Beijing University of Technology, Beijing, China, and the Ph.D. degree (with the honor of Hashimoto Prize) in computer engineering from the New Jersey Institute of Technology, Newark, NJ, USA, in 2006.

She is currently an Associate Professor in the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA, USA. Her recent research interests include trusted computing, computing and communication architectures for chip multiprocessors, reconfigurable computing, VLSI design, and embedded systems.