International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Sys 6, No.5 (Sep-2017)
http://dx.doi.org/10.12785/ijcds/060501

# Networks-on-Chip Interfacing:
## NoC-based Systems Networking and Inter-NoC Communication

**Ahmed S. Hassan[1], Ahmed A. Morgan[2] and M. Watheq El-Kharashi[1,3]**

[1]*Department of Computer and Systems Engineering, Ain Shams University, Cairo, Egypt*
[2]*Department of Computer Engineering, Cairo University, Giza, Egypt*
[3]*Electrical and Computer Engineering Department, University of Victoria, Victoria, Canada*

*Received 17th Jun. 2017, Revised 12 Jul. 2017, Accepted 20th Aug. 2017, Published 1st Sep. 2017*

**Abstract:** Many- and multi-core Networks-on-Chip (NoC) systems have a large spectrum of applications, and one of these applications is High Performance Computing (HPC). In HPC application, system scalability is one of the important features of a given HPC platform. Two of the factors governing scalability are the interfacing link speed and the ease of deploying additional processing nodes. For NoC-based systems, scalability has two meanings; intra-NoC scalability, which includes partitioning and clustering Processing Elements (PEs) to achieve better performance, and inter-NoC scalability, which includes interfacing with other NoC-based systems via inter-NoC links and creating a newtrok of multiple NoC-based systems. In this paper, we investigate the case of inter-NoC communication between a network of NoC-based systems. By using our simulator, we found out that treating a network of NoC-based systems as a generic case of interfacing yields non-optimal performance. To target the inter-NoC communication performance, we introduce *NoC*$^2$, an NoC-based system with an Ethernet communication manager that provides better management for inter-NoC traffic. Current implementations of inter-NoC traffic management handle inter-NoC traffic using software running on top of a dedicated PE, whereas *NoC*$^2$ is designed so that the software running on cores within the NoC is completely abstracted from inter-NoC traffic management. Experiments show that this kind of abstraction has better inter-NoC communication performance over generic NoC interfaced via Ethernet. Results suggest that *NoC*$^2$ would perform better when it comes to scaling NoC-based systems by interfacing more NoC-based hardware.

**Keywords:** Inter-NoC communication; *NoC*$^2$; NoC Ethernet; NoC interfacing; NoC network; NoC scalability.

## 1. Introduction

In high performance clustered computing applications, system designers need to take advantage of system scalability. Ideal systems would be able to scale up in a homogeneous way, ideally by just plugging-in additional computation units. One way of scaling Networks-on-Chip (NoC)-based systems is clustering; by grouping Processing Elements (PEs), either statically at topology generation, or dynamically at runtime. Cluster groups allow adding resources without having to re-design the whole system. Software running on PEs should be isolated from managing traffic between clusters and clusters should be self-organized as much as possible. NoC-based systems are scalable in nature, but when it comes to interconnecting NoC-based systems, NoC scalability becomes challenging. For instance, to achieve optimal performance, NoC-based systems should be aware of being interconnected to a new NoC-based system. In this case, running software should be aware of the new NoC and be able to configure the NoC-based system for optimum traffic handling. Having the software aware of the underlying NoC is not optimal. It would cost additional overhead for NoC management and make the software prone to portability issues, if it is intended to run on different NoC-based systems or even non-NoC systems.

In this paper, we introduce building a network of NoC-based systems (*NoC*$^2$). *NoC*$^2$ is an inter-NoC communication methodology that realizes the inter-NoC communication as normal NoC traffic of flits within a network of NoC-based systems. *NoC*$^2$ addresses the challenge of traffic communication between PEs in different NoC-based systems without requiring changes in the software running on those PEs to control this kind of traffic. Additionally, *NoC*$^2$ considers the communication peripheral as a part of the NoC topology, not just a secondary peripheral attached to a generic PE. From this sense, Network Interfaces (NIs) are aware of this secondary peripheral and therefore can communicate directly with its controller instead of routing

*E-mail address: ahmedsayed.elaraby@gmail.com, ahmorgan@eng.cu.edu.eg, watheq.elkharashi@eng.asu.edu.eg*

flits around to the PE with the controller attached. We adopt a distributed buffering mechanism in the First-In-First-Out (FIFO) queues between the NIs and the communication peripheral.

We claim, and the results support our claim, that distributed FIFOs and direct communication between the NIs and the communication peripheral are a better option for the case of interconnecting nodes in a network of NoC-based systems, compared to the generic NoC, using centralized FIFOs and routed communication.

Standard NoC-based systems, as shown in Figure 1, consist of PEs attached to NIs. NIs are interconnected using the NoC routers. Traffic generated by a PE is sent to its corresponding NI, which prepares the flits and send them to the router. Routers inspect the flits and take a decision where they should route them; to a nearby router or to the local NI port.

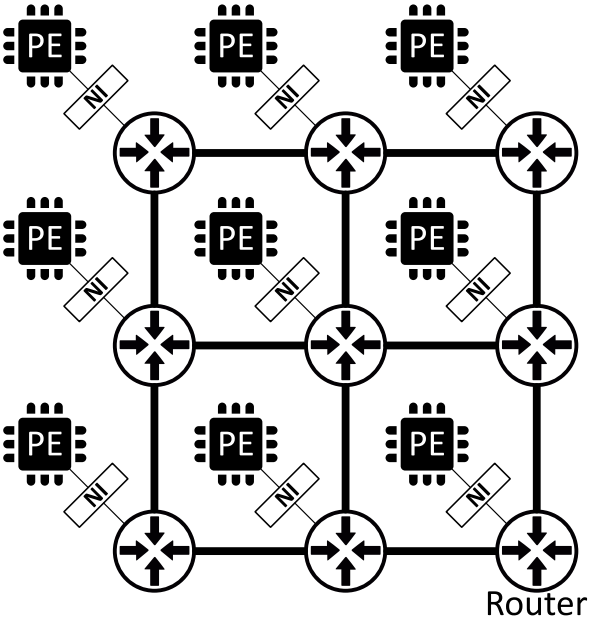


Figure 1. Standard NoC components.

Most of the NoC literature refers to NoC traffic to only be the intra-NoC traffic; that is the traffic between PEs on the same NoC-based system. Any traffic that goes outside the NoC-based system is considered a non-NoC traffic, even if it would go to another NoC-based system. Such mindset would limit the scalability of NoC traffic communication in industrial domains. In high-performance computing applications, a system consists of several computation nodes interconnected with each other. Those nodes themselves could be multi- or many-core systems, on which the PEs form a NoC-based system. Connection links between nodes should provide high-speed communication, so that delays due to communication do not degrade performance. High-speed interconnection technologies, like PCI [1] and RapidIO [2], are often used for on-board communication, like interconnecting processors [3], [4]. Ethernet [5] and Infiniband [6], on the other hand, are often used for off-board communication, like interconnecting computer servers [7], [8].

The rest of the paper is organized as follows. Section 2 reviews the related work, Section 3 discusses NoC Interfacing, Section 4 presents our $NoC^2$ model. Sections 5 and 6 shows our experiment and results. The paper is concluded in section 7.

## 2. Related Work

The majority of NoC scalability research focuses on clustering and partitioning of nodes on the same chip, either statically, at the topology generation, or dynamically at runtime [9].

An example of static clustering techniques, Clustered NoC (C-NoC) was introduced by Seifi *et al.* [10]. C-NoC groups nodes, based on their communication density, and connect these nodes to the same router. To do this, they proposed a modified router architecture, based on Hermes NoC [11]. The modified router partitions the system to local and remote ports. Local ports connect nodes with high communication volume, while remote ports is connected to other routers. Figure 2 shows how C-NoC clusters PEs and group local PEs into the same router via local links.

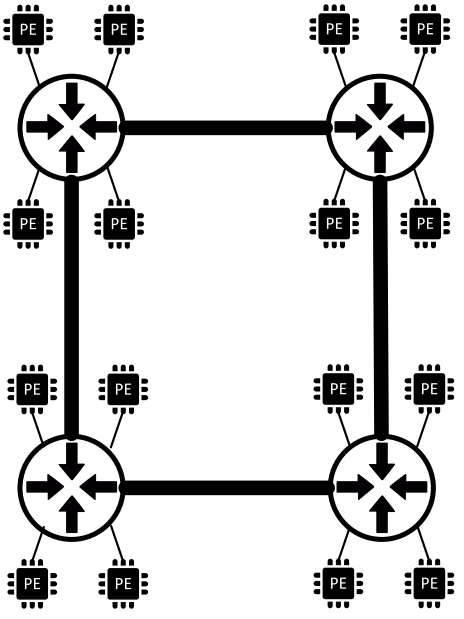


Figure 2. Clustering in C-NoC [10].

Yuan *et al.* [12] proposed a NoC clustering architecture for Graphics Processing Unit (GPU)-based data parallel architecture. Their proposal groups compute cores and GPU memory controller into clusters, where each cluster consists of a group of cores and one memory controller. Their study shows significant improvement over the nun-clustered case.

Hamid *et al.* [13] proposed a static clustering for Multi-Core Multi-Cluster Architecture (MCMCA). In this proposal, multi-core chips are grouped into cluster, where multiple clusters are interconnected via Multi-Cluster Network (MCN). They proposed two different networks for cluster communication, Intra-Cluster Network (ACN) and Inter-Cluster Network (ECN).

Castilhos *et al.* introduced a dynamic approach for NoC clustering [14]. The study proposes distributed resource management in NoC-Based Multi-Processor System-on-Chip (MPSoC) using a clustering method that supports modification of the cluster size at runtime. The work is focused on homogeneous MPSoC architecture interconnected through 2D NoC. The MPSoC is divided into equally-sized clusters, where each cluster has one local cluster manager (LMP) and one cluster has a global master (GMP). The LMP is responsible for control the cluster, executing functions such as monitoring, task mapping, deadlines verifications and communication with other LMPs and the GMP. The GMP has all functions of the LMP, and functions related to the overall system management such as controlling the available resources in each cluster. The dynamic management allows a cluster to send a loan request to borrow resources from neighbor clusters, if its resources do not meet the processing requirements.

Other partitioning techniques aim at performance optimization, like the work of Morgan *et al.* [15]–[19], or overall system performance modeling, like the model proposed by Elmiligi *et al.* [20]–[24].

There is little research about NoC-based systems interfacing to off-chip systems. Wasicek [25] proposed NoC-based Ethernet gateway to interface MPSoC systems. In Wasicek's proposal, inter-MPSoC communication is achieved by implementing the gateway logic on a dedicated NoC PE, which had an Ethernet controller attached to it. The gateway treats NoC traffic and inter-MPSoC traffic differently and it had to convert message formats between NoC network and inter-MPSoC network. In our proposal, we only deal with NoC flit traffic so there is no need to implement a gateway. We also adopt a distributed FIFO mechanism to avoid creating hotspots around the interfacing peripheral.

We found no research in the literature that treats inter-NoC traffic as genuine NoC traffic. We could not even find a NoC simulator that supports clustering or inter-NoC traffic. In order to simulate our system, we added support for clustering simulation to one of the NoC simulators. In our previous work, we extended a NoC simulator, NoCTweak [26], so that it would support cluster simulation [27], [28].

## 3. NoC Interfacing

By the term NoC Interfacing, we mean the case where one or more communication peripherals are attached to one or more nodes on a given NoC-based system. Those communication peripherals are connected to other external systems. External systems can be computational modules, sensors, or any other unit. One of the possibilities for NoC interfacing is to interface with another NoC-based system. For this case, we can roughly divide communication type into two categories:

1) **Generic communication:** This is the normal case, where traffic has no special meaning to the underlying NoC. In this case, the PE with the communication peripheral, talks to another PE in a remote NoC-based system.
2) **Inter-NoC communication:** In this case, communication peripherals are not necessarily attached to the communicating PEs. When two PEs, in two different NoC-based systems, want to communicate they must first send the data to the edge PE in their system, which has the communication peripheral. The edge PE sends the data to the other edge PE in the remote system and this PE sends the data through the NoC to the original destination PE.

In this section, we focus on the Inter-NoC communication.

### A. Inter-NoC Communication

Inter-NoC communication primarily addresses NoC scalability. Scalability is defined as the ability to expand the NoC-based system processing power. The simplest form of NoC scalability is to ditch the old system and design a brand new system with larger size and more resources. This scalability form is not practical nor industry friendly, as it requires replacing the hardware, coming with a cost on both the replacement process and the hardware itself. Another form of scalability is to get another NoC-based system and interface it to the original NoC-based system. By taking advantage of high-speed communication peripherals, we can construct a network of NoC-based systems. This latter form promotes for a more practical NoC scalability option for industrial applications.

Inter-NoC traffic is considered a normal NoC traffic, consisting of NoC flits. Inter-NoC traffic can be handled by the edge PE. This is the simplest solution that is entirely based on software. We will refer to this case as Generic NoC Interfacing. Generic NoC interfacing handles two situations:

- Sending traffic to a remote NoC-based system: A software running on top of PEs controls whether a transmitted flit should go to another PE within the same NoC-based system or should go to a remote one. Based on this, the "send data to local PE" routine is issued, or "send data to edge PE" routine is issued for the other case.
- Receiving data from a remote NoC-based system: The edge PE then inspects the received flit and transmits it to the appropriate edge PE in the other remote NoC-based system. Upon receiving a flit, edge routers investigate the intended destination within its local NoC-based system and then issue a normal "send data to local PE" routine.

To distinguish between the final destination PE in the remote system and the intermediate destination edge PE in the local system, the flit has to be modified to include the additional destination options.

### B. Case Study

By using the tool proposed in our previous study [27], [28], we managed to simulate the case of inter-NoC traffic. The aim of this simulation was to evaluate the performance of the intuitive inter-NoC communication technique; routing the outbound traffic to the edge PEs.

#### 1) Inter-NoC Simulation Setup

Originally, our tool was designed to simulate NoC clustering. The tool provides some additional features added to NoCTweak, mainly to serve the clustering simulation. Clustering simulation is achieved by creating multiple instances of the NoC and creating links between these instances. The tool provides means to configure the latency factor of the inter-cluster links.

By using this feature, inter-NoC traffic simulation can be performed by configuring the inter-cluster links to have the delay ratio between intra-NoC links and the inter-NoC links. For example, if two NoC-based systems are connected by an Ethernet link and an Ethernet packet takes 256 cycles to get from source to destination, whereas intra-NoC links take 16 cycles, then the inter-cluster configuration parameter should be configured to 256/16 = 16.

Figure 3 shows the NoC network simulated. Simulation environment is configured for 4 NoC-based systems grouped in 2$x$2 cluster. Each NoC-based systems contains 4 PEs connected as a 2$x$2 mesh topology. There is one inter-cluster link between each adjacent NoC-based system. Each NoC-based system is configured with 4 Virtual Channels (VCs) and packet length of 1 flit. Inter-cluster weight is configured to 16. We ran the simulation multiple times with different Flit Injection Rate (FIR) values, ranging from 0.1 to 1.0 with step of 0.1.

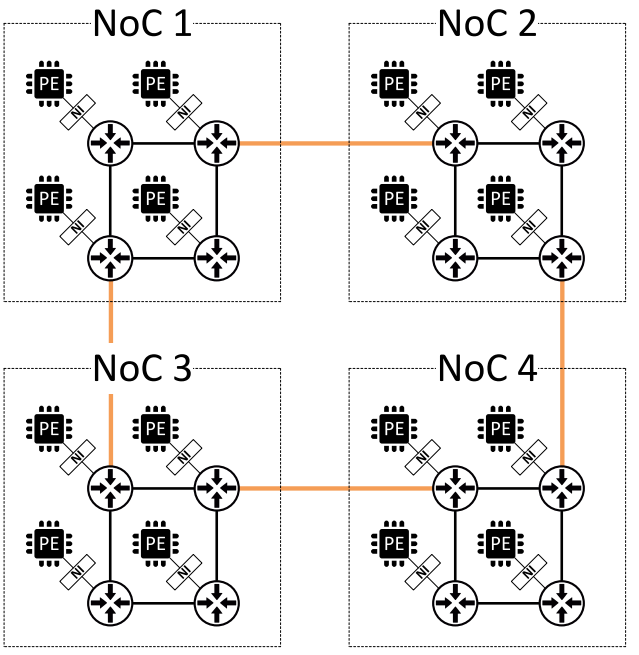


Figure 3. 2$x$2 cluster of 4 2$x$2 NoC-based systems.

#### 2) Simulation Results

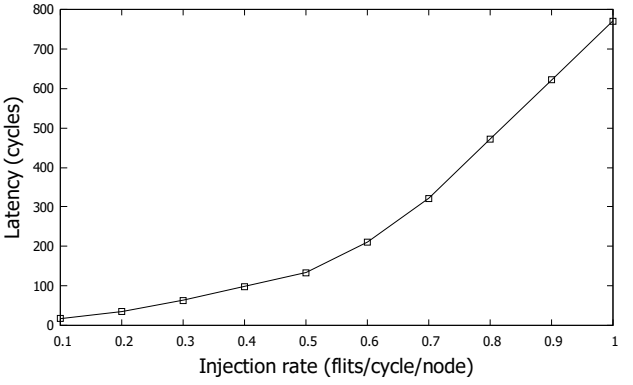


Figure 4. Latency measurements for simulation of 2$x$2 cluster of 4 2$x$2 NoC-based systems.

Figure 4 shows the latency results collected after running the simulation. Results show exponential increase in flit latency as the FIR increases. This suggests that connecting NoC-based systems via a single link connected to a single node in the NoC-based system is not optimal. This results encouraged us to propose a different approach for interconnecting NoC-based systems.

## 4. $NoC^2$ Model

The concept behind $NoC^2$, as shown in Figure 5, is to let NIs take decisions whether to send a flit to the routers,

or to ***I***nter-No***C FIFO*** (ICFIFO). ICFIFO is a controller with two FIFOs, Tx for transmitting and Rx for receiving. ICFIFO is connected to a network device, e.g., Ethernet. Whenever ICFIFO Tx FIFO is not empty, a transmission is issued to the network device. Whenever a message is received by the network device, it gets passed to ICFIFO's Rx FIFO, which in turn gets consumed by the appropriate NI.

### A. Processing Element and Network Interface

Compared to a generic NoC implementation, the PE will not change, nor the software running on top of it. The NI is modified to decide where to forward the flit, as mentioned previously. Each NI has a newly-added special function register, PE Total Number (*PETN*), which holds the total number of PEs in the whole system. The total number of PEs would be the total number of PEs in all NoC-based systems in the network. The *PETN* register gets updated whenever the interconnected system changes, by a special control message sent from the ICFIFO controller to each NI.

### B. Inter-NoC FIFO

ICFIFO is responsible for queuing flits and triggering Ethernet transmission for egress traffic, receiving flits via Ethernet, and then passing them to the corresponding NI for ingress traffic. Figure 6 shows the components of the ICFIFO:

- Controller, which executes the core logic of ICFIFO and controls the communication peripheral.
- Group of FIFOs, which are used to communicate with the NIs in the system.
- Communication Peripheral Controller (CPC), which is used to connect the system to other remote systems.
- Register File (RF), a group of registers to keep the status of the ICFIFO, like the *PETN* register.

In ICFIFO, we did not just increase the number of FIFOs. We actually relocated them in a distributed way. A normal NI in an NoC-based system has multiple buffers configured as VCs. What we did in ICFIFO is to relocate some of those buffers, outside each NI/router interface, to multiple NI/ICFIFO FIFOs. In this way, each inter-NoC flit has a deterministic path of a single direct hop from NI to its associated FIFO in the ICFIFO, instead of routing it around the NoC, thus separating the inter-NoC traffic from the intra-NoC traffic management. The overall number of FIFOs in the system can remain unchanged, i.e., if we had a $2x2$ NoC and each NI/router interface has FIFOs, and we make ICFIFO with 8 FIFOs and shorten the NI/router to 2 FIFOs. We will still have an overall of 16 FIFOs, but FIFOs are distributed in a more friendly way for inter-NoC traffic in a network of NoC-based systems.

#### 1) Initialization Sequence

When an Ethernet MAC is connected to another NoC-based system, the ICFIFO detects that and triggers a transmission with data containing the current number of PEs in the interconnected system. For a single NoC-based system, this number is initialized with the number of PEs in this system. The ICFIFO expects a received message from any other system, containing also the number of PEs in that system. ICFIFO in each of the interconnected systems updates its internal value, then broadcasts a special control message to NIs in its local system. Each NI updates its *PETN* register upon receiving the control message from the ICFIFO. Herein, we should re-emphasize that this message does not get to the core attached to the NI. Each system sends control message to its neighboring systems and waits for acknowledgment from them, thus completing the handshake procedure and the interconnected systems are ready for communication.

#### 2) Egress Traffic

In case of a PE transmitting traffic, the NI inspects the flit header. If the destination ID does not match any of the local NoC nodes ID, which is held in a register inside each NI, the NI writes the flit into the Tx FIFO associated with this particular NI. Whenever a Tx FIFO is not empty, the ICFIFO dequeues it into an Ethernet frame. After each dequeue, Ethernet frame transmission is triggered.

Figure 7 shows how the NI behaves upon triggering flit transmission. PE #3 sends two flits, one for PE #8 and the other for PE #10. For both flits, the NI inspects the destination field. For the first flit, the destination matches one of the local PEs, so the NI enqueues this flit into the NoC router. The second flit is intended to a PE that is not a part of the local NoC, so the NI enqueue this flit into its associated ICFIFO Tx FIFO. The ICFIFO's controller then triggers the CPC for packet transmission to the other NoC-based system.

#### 3) Ingress Traffic

In case of receiving a traffic via Ethernet, the ICFIFO inspects the received message, then enqueues it into the corresponding Rx FIFO. Whenever an Rx FIFO is not empty, the NI interrupts the core within the node and the core would dequeue the FIFO and get the received flit.

Figure 8 shows the case of ingress traffic. Flits received by the CPC are inspected by the ICFIFO's controller, and then enqueued to the Rx FIFO corresponding to the destination PE.

## 5. Evaluation

We performed an experiment to test $NoC^2$ performance in handling inter-NoC traffic. We compared $NoC^2$ performance against another generic NoC-based system.

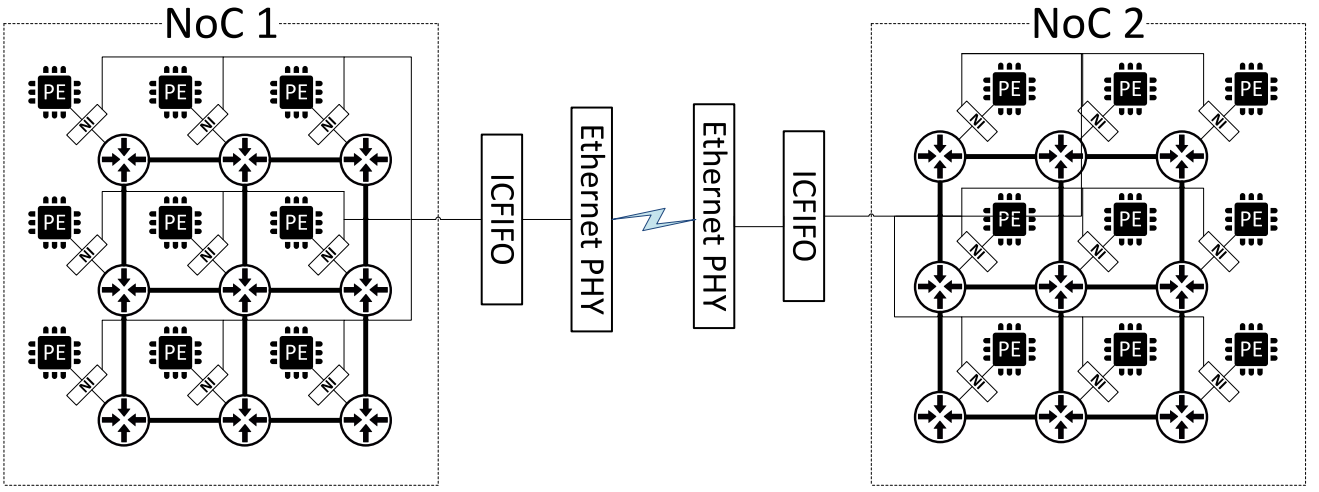


Figure 5. $NoC^2$ interconnection model.

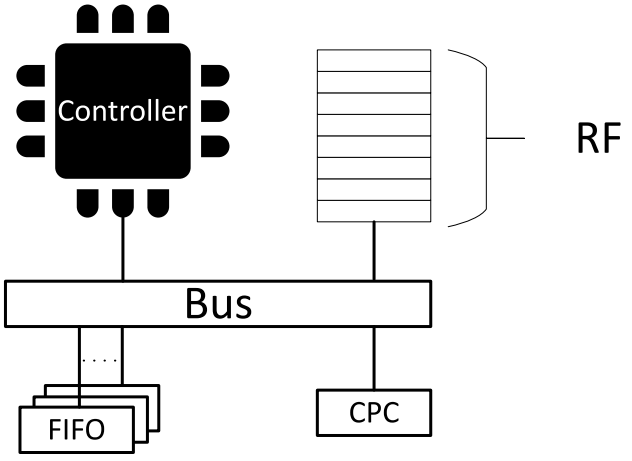


Figure 6. ICFIFO internal structure in $NoC^2$.

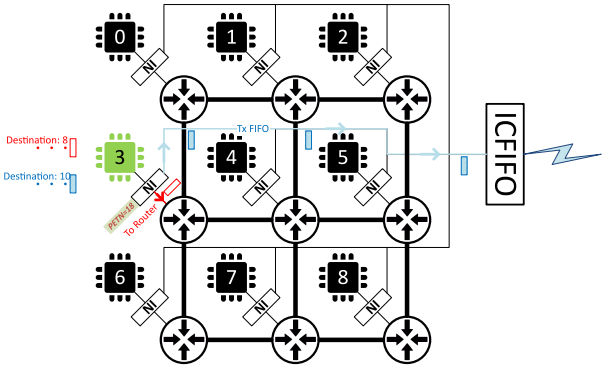


Figure 7. An example of egress traffic path.

### A. Experiment

Our experiment involved two NoC-based systems, NoC0 and NoC1, which were interconnected via Ethernet. One end is a NoC-based system running on Artix-7 [29] FPGA and the other end is a simulation software running on PC. Each PE CPU is running at 16 MHz.

### B. Experiment Environment

In our experiment, both NoC-based systems have 2x3 2D mesh topologies, XY routing, and a 4-byte payload. The software running on each PE on both NoC-based systems sends a message to other three PEs. The PE destinations are randomly selected from the PE pool across the two NoC-based systems. PE ID to grid mapping follows this equation:

$$ID = (NoC\_id * X_{dim} * Y_{dim}) + (x + y * Y_{dim}) \quad (1)$$

In Equation (1), $NoC\_id$ is the ID of the NoC-based system in the network, $X_{dim}$ and $X_{dim}$ are the size of the NoC-based system with ID = $NoC\_id$, and $x$ and $y$ are location of the PE in this particular NoC-based system.

The software running on PEs controls the Packet Injection Rate (PIR). The PIR rate represents how many packets a PE generates per second. In our test, each packet consists of three flits. So, each PE triggers three NI transmissions per packet. PIR followed the normal distribution. Therefore, half of the PEs in any system are communicating to PEs in the other NoC-based system via inter-NoC traffic at any given time. We tested with PIR rates at 1, 50, 100, 200, 400, 500, and 1000 packet/second per PE node.

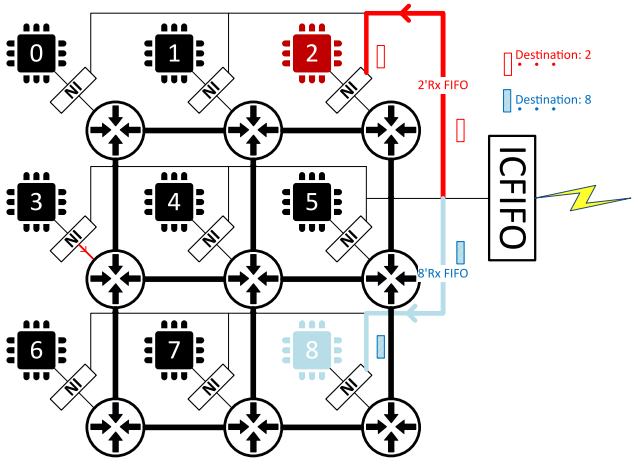


Figure 8. An example of ingress traffic path.

Ethernet packets are captured and analyzed using Wireshark [30]. Wireshark is a network sniffing and analysis tool which we used to asses the Ethernet link throughput and packet latency. Wireshark statistics provide various information about the captured packets, like average packets captured per second (Pbs), total number of bytes captured and average throughput in Megabit per second (Mbps). Figure 9 shows an example of statistics generated by Wireshark. In this example, it is shown that the average Pbs is 1313. We use the Pbs information to asses ICFIFO performance, by plotting it against PIR.

### C. $NoC^2$-based System Implementation

This section explains how we implemented $NoC^2$ components on FPGA: the PE node and the ICFIFO node.

#### 1) PE Node Implementation

The PE node implementation is based on the open source Wishbone bus [31]. The Wishbone bus interconnects an AEMB processor [32], a single-port RAM block, a general-purpose input/output (GPIO) block, an interrupt controller (Int.), and an NI block. Figure 10 shows a diagram of the PE node implementation.

#### 2) NI Implementation

Besides the basic NI functionality, we added the *PETN* register so that each NI in the system can decide if a flit is intended for intra-NoC or inter-NoC.

#### 3) ICFIFO Node Implementation

The ICFIFO node implementation is based on Wishbone bus as well. It contains 64-byte Dual-Port RAM (DP RAM), Ethernet MAC (Eth), interrupt controller, and an AEMB controller. The DP RAM is used as a storage for the FIFO, while the controller does both controlling the Ethernet MAC and implementing the ICFIFO logic. We did not implement the ICFIFO controller as ASIC, rather we executed the controller logic on top of AEMB processor. We did this because we just wanted to evaluate the effect of ICFIFO on inter-NoC traffic, rather than provide optimized implementation for it. In the current design, each NI in the system has a dedicated DP RAM, implementing bidirectional FIFO. Figure 11 shows a diagram of the ICFIFO node implementation.

### D. The Generic NoC

The generic NoC-based system has the same NoC architecture as $NoC^2$, but with normal NI. The payload is doubled to 8 bytes. Payload size had to increase so that an additional field, remote destination, can be added to the original flit, and in our NoC design the NI handles flit size as multiple of 4 bytes only. One of the PE nodes in this generic NoC has Ethernet MAC, and the software running on this PE is responsible for interfacing with other NoC-based systems. Other PEs wishing to communicate with remote NoC-based system should direct their traffic to this particular PE as well, i.e., software running on different PEs is aware of the NoC beneath it. The remote destination information is checked by the software running on the node with Ethernet and forwards this information to the other NoC-based system. That is why we had to increase the payload size. This case is different from the case of $NoC^2$, where handling of inter- and intra-NoC traffic is handled by NIs and the ICFIFO without any required intervention from the PEs.

Pseudocode for the software running on the PE with Ethernet is shown in Algorithm 1. The algorithm shows how the generic NoC would behave in three cases:

- Sending inter- or intra-NoC message, lines 4–8. The PE examines the destination ID *dst*. If *dst* does not belong to this NoC, then it gets transmitted over Ethernet. If, on the other hand, *dst* belongs to this NoC, then the PE will transmit the message over NI to the destination PE.
- Receiving intra-NoC message via NI, lines 9–14. The PE examines the destination ID *dst*. If the received *dst* is the same as this PE ID, *peId*, then the PE extracts the message and continue its normal processing with the message contents, e.g., sends data to application. If, on the other hand, the message is not intended to this PE, it will get transmitted via Ethernet to its intended destination.
- Receiving inter-NoC message via Ethernet, line 15–19. The PE examines the destination ID *dst*. If the received *dst* is the same as this PE ID, *peId*, then the PE extracts the message and continues its normal

| Traffic | Captured | Displayed | Displayed % | Marked | Marked % |
|---|---|---|---|---|---|
| Packets | 66262 | 66262 | 100.000% | 0 | 0.000% |
| Between first and last packet | 50.452 sec | 50.452 sec | | | |
| Avg. packets/sec | 1313.370 | 1313.370 | | | |
| Avg. packet size | 78.000 bytes | 78.000 bytes | | | |
| Bytes | 5168436 | 5168436 | 100.000% | 0 | 0.000% |
| Avg. bytes/sec | 102442.833 | 102442.833 | | | |
| Avg. MBit/sec | 0.820 | 0.820 | | | |

Figure 9. An example of Wireshark packet capture statistics.

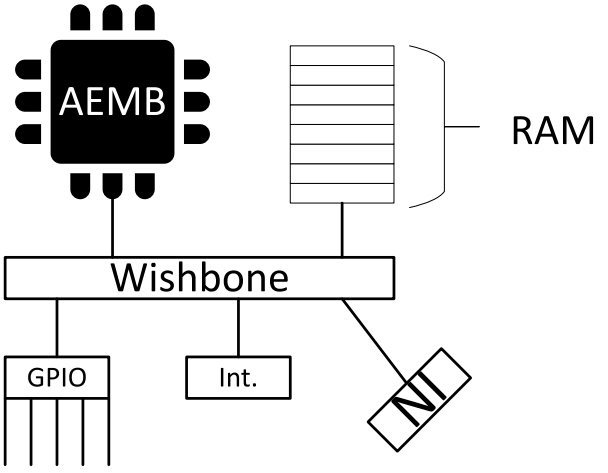


Figure 10. PE tile implementation for $NoC^2$

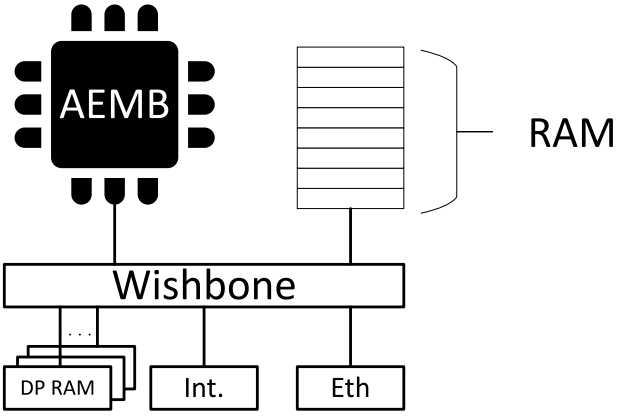


Figure 11. ICFIFO in $NoC^2$.

processing with the message contents, e.g., sends data to application. If, on the other hand, the message is not intended to this PE, it will get transmitted via NI to its intended destination within the same NoC.

## 6. Results

Because $NoC^2$ is designed for inter-NoC communication, we oriented our analysis towards performance related to inter-NoC traffic, other than intra-NoC traffic. Thus, we were more interested in inter-NoC Ethernet packets throughput and latency.

### A. Throughput and Latency

Here, we measured the Ethernet frame throughput and latency.

Throughput results, shown in Figure 12, show that generic NoC throughput degrades more rapidly than $NoC^2$ as the PIR increases. In comparison, there is significant improvement in the throughput for $NoC^2$. $NoC^2$ shows more stable throughput, where it increases to a higher rate, then it saturates but does not degrade. The reasons for this improvement are:

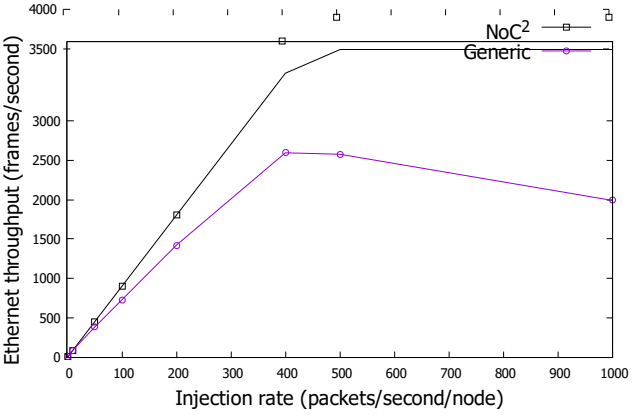


Figure 12. Inter-NoC packet throughput versus PIR.

- As the PIR increases the Ethernet node in generic NoC becomes a hotspot, and the rate of congestions

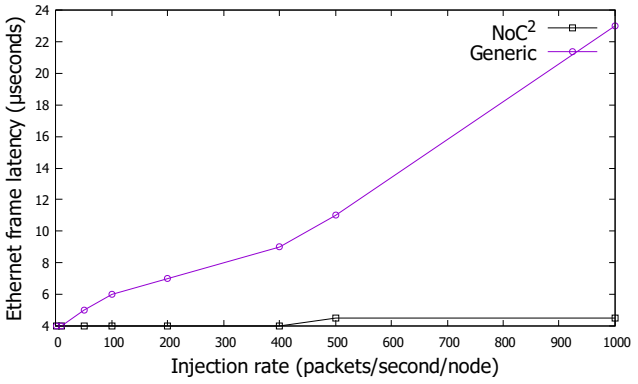


Figure 13. Inter-NoC packet latency versus PIR.

**Algorithm 1** Generic NoC: Sending and receiving by PE with Ethernet.

```
1: dst ← Destination PE
2: maxId ← Maximum PE ID in the NoC
3: peId ← PE own ID
4: procedure SendMsg
5:     if dst > maxId then
6:         EthernetTransmit(to: dst)
7:     else
8:         NiTransmit(to: dst)
9: procedure RecvMsgFromNi
10:    if dst == peId then
11:        Deliver message to application ...   ▷ Message was intended to this PE
12:    else
13:        if dst > maxId then
14:            EthernetTransmit(to: dst)
15: procedure RecvMsgFromEth
16:    if dst == peId then
17:        Deliver message to application ...   ▷ Message was intended to this PE
18:    else
19:        NiTransmit(to: dst)
```

and stalls increases. The NI attached to the Ethernet MAC gets more traffic load than other NIs in the system, and that is because every node, which needs to communicate with the remote system, routes its traffic to the Ethernet NI. So, this NI buffers get overloaded and stall, and hence the hotspot is generated.

- In contrast, $NoC^2$ was able to scale up with the transmission rate. No hotspot nodes were formed and measurements show that each transmission job consumed 64 cycles constantly through different PIR rates. By having ICFIFO, the system did not exhibit stalls when subjected to a traffic load similar to the generic NoC case.

Figure 13 shows that generic NoC had an increasing average latency due to stalls, while $NoC^2$ had lower latency of 4 to 5 $\mu$sec per frame. The reason behind this latency improvement is that there were no hotspots occurred on the ICFIFO buffers. On the other hand in generic NoC, the PE node with Ethernet controller is overwhelmed with flits, that it caused a bottle neck hotspot.

## 7. Conclusion and Future Work

In this paper, we have studied interconnecting NoC-based systems. We provided a simulation usecase for generic interconnection of NoC-based systems, and showed that the traditional inter-NoC communication approach which treats PEs, with communication peripherals, as any other generic PE, is not optimal for interconnecting NoC-based systems. By having communication peripherals controlled by PEs, we limit the software portability, as the inter-NoC traffic has to be controlled by software. We also get a degraded performance, due to hotspots created on the interfacing PE.

We have introduced $NoC^2$ to address the case of inter-NoC traffic, where NoC-based systems are connected via Ethernet. Our design showed significant performance improvement over generic NoC interconnection. Results showed higher throughput and stable latency for the inter-NoC traffic when it is routed within a system interconnected with our $NoC^2$ approach in comparison to another generic NoC system.

In the future, we may use other high-speed technologies oriented towards chip-to-chip communication, like Serial RapidIO, by adding support for those technologies in the CPC module. We may also implement the ICFIFO as ASIC.

### Acknowledgment

This paper is a significant extension and update of a paper that appeared in the proceedings of the 4th Workshop on Design and Performance of Networks on Chip (DPNoC 2017) in conjunction with the The 12th International Conference on Future Networks and Communications (FNC 2017) [33]. All trademarks ™ and registered trademarks ® mentioned, cited, or referenced in this document remain the property of their respective owners.

## References


[1] "PCI-SIG," https://pcisig.com, [Last visited 2017-09-01].

[2] "RapidIO," http://www.rapidio.org, [Last visited 2017-09-01].

[3] R. Bittner, E. Ruf, and A. Forin, "Direct GPU/FPGA communication via PCI express," *Cluster Computing*, vol. 17, no. 2, pp. 339–348, 2014.

[4] X.-y. Huang, H.-b. Su, Q.-z. Wu, and W. Wu, "Multi-Processor Parallel System Based on High-Speed Serial Transceiver," in *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, vol. 1, 2010, pp. 178–181.

[5] "IEEE 802.3 ETHERNET WORKING GROUP," http://www.ieee802.org/3, [Last visited 2017-09-01].

[6] "InfiniBand Trade Association," http://www.infinibandta.org, [Last visited 2017-09-01].

[7] J. Zhang, X. Lu, and D. K. Panda, "High Performance MPI Library for Container-Based HPC Cloud on InfiniBand Clusters," in *Parallel Processing (ICPP), 2016 45th International Conference on*, 2016, pp. 268–277.

[8] E. Gamess and H. Ortiz-Zuazaga, "Evaluation of Point-to-Point Network Performance of HPC Clusters at the Level of UDP, TCP, and MPI," in *IV Simposio Científico y Tecnológico en Computación, Caracas, Venezuela*, 2016.

[9] F. Gebali, H. Elmiligi, and M. W. El-Kharashi, *Networks-on-chips: theory and practice*. CRC press, 2011.

[10] M. R. Seifi and M. Eshghi, "A clustered NoC in group communication," in *TENCON 2008-2008 IEEE Region 10 Conference*. Hyderabad, India: IEEE, 2008.

[11] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *INTEGRATION, the VLSI journal*, vol. 38, no. 1, pp. 69–93, 2004.

[12] W. Yuan, R. Boyapati, L. Wang, H. Jang, Y. Jin, K. H. Yum, and E. J. Kim, "Intra-clustering: Accelerating on-chip communication for data parallel architectures," in *2015 International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW)*. IEEE, Oct 2015, pp. 55–60.

[13] N. Hamid, R. J. Walters, and G. Wills, "Understanding the impact of the interconnection network performance of multi-core cluster architectures." *Journal of Computers*, vol. 11, no. 2, pp. 132–139, 2016.

[14] G. Castilhos, M. Mandelli, G. Madalozzo, and F. Moraes, "Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes," in *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Natal, Brazil: IEEE, Aug 2013, pp. 153–158.

[15] A. A. Morgan, H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Multi-objective optimization for networks-on-chip architectures using genetic algorithms," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 3725–3728.

[16] A. A. Morgan, H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Networks-on-chip architecture customization using network partitioning: A system-level performance evaluation," *International Journal of Computing and Digital Systems*, vol. 4, no. 1, pp. 19–31, Jan 2015.

[17] A. A. Morgan, H. Elmiligi, F. Gebali, and M. W. El-Kharashi, "Unified multi-objective mapping and architecture customisation of networks-on-chip," *IET Computers & Digital Techniques*, vol. 7, no. 6, pp. 282–293, 2013.

[18] A. A. Morgan, H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Area-aware topology generation for application-specific networks-on-chip using network partitioning," in *2009 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE, 2009, pp. 979–984.

[19] A. Morgan, H. Elmiligi, M. El-Kharashi, and F. Gebali, "Bio-inspired NoC architecture optimization," in *Autonomic Networking-on-Chip: Bio-Inspired Specification, Development, and Verification*, P. Cong-Vinh, Ed. CRC Press, 2012, pp. 21–45.

[20] H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Modeling and implementation of an output-queuing router for networks-on-chips," pp. 241–248, 2007.

[21] H. Elmiligi, A. A. Morgan, M. W. El-Kharashi, and F. Gebali, "A topology-based design methodology for networks-on-chip applications," in *Proceedings of the Second IEEE International Design and Test Workshop (IDT 2007)*. IEEE, 2007, pp. 61–65.

[22] H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Power consumption of 3D networks-on-chips: Modeling and optimization," *Microprocessors and Microsystems*, vol. 37, no. 6, pp. 530–543, 2013.

[23] H. Elmiligi, A. A. Morgan, M. W. El-Kharashi, and F. Gebali, "Power optimization for application-specific networks-on-chips: A topology-based approach," *Microprocessors and Microsystems*, vol. 33, no. 5, pp. 343–355, 2009.

[24] H. Elmiligi, A. A. Morgan, M. W. El-Kharashi, and F. Gebali, "A reliability-aware design methodology for networks-on-chip applications," in *Proceedings of the 2009 Fourth IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'09)*, H. El-Tahawy, M. Abadir, A. Jerraya, and A. Salem, Eds., Cairo, Egypt, 2009, pp. 107–112.

[25] A. Wasicek, "Embedding complex embedded systems in large ethernet–based networks," *Network*, vol. 1, no. C2, p. C3, 2011.

[26] A. Tran and B. Baas, "NoCTweak: A highly parameterizable simulator for early exploration of performance and energy efficiency of networks on-chip," *VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep. ECE-VCL-2012-2*, 2012.

[27] A. S. Hassan, A. A. Morgan, and M. W. El-Kharashi, "An enhanced network-on-chip simulation for cluster-based routing," *Procedia Computer Science*, vol. 94, pp. 410–417, 2016.

[28] A. S. Hassan, A. A. Morgan, and M. W. El-Kharashi, "Clustered networks-on-chip: Simulation and performance evaluation," *International Journal of Computing and Digital Systems*, vol. 6, no. 2, pp. 51–61, Mar. 2017.

[29] "Artix-7," https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html, [Last visited 2017-09-01].

[30] "Wireshark," https://www.wireshark.org, [Last visited 2017-09-01].

[31] "Wishbone," http://opencores.org/opencores,wishbone, [Last visited 2017-09-01].

[32] "aeMB," https://opencores.org/project,aemb, [Last visited 2017-09-01].

[33] A. S. Hassan, A. A. Morgan, and M. W. El-Kharashi, "Introducing $NoC^2$: interconnecting noc-based systems through ethernet," *International Journal of Computing and Digital Systems*, vol. 6, no. 2, pp. 51–61, Mar. 2017.

**Ahmed S. Hassan** Ahmed S. Hassan received B.Sc. degree in systems and biomedical engineering, Cairo University, Egypt, in 2011. He is an embedded software developer, specialized in multicore architecture, wireless connectivity, and automotive Ethernet. Currently an M.Sc. candidate at Ain Shams University, Cairo, working on many-core Systems-on-Chip (SoC) analysis and design.

**Ahmed A. Morgan** Ahmed A. Morgan received the Ph.D.degree from the University of Victoria, Victoria, BC, Canada, in 2011, and the B.Sc. degree (first class honors) and the M.Sc. degree from the Faculty of Engineering at Shoubra, Benha University, Egypt in 2000 and 2005, respectively. He got a Diploma in Electronic Design Automation (EDA) and VLSI Design from the Information Technology Institute (ITI), Cairo, Egypt in 2002. He is an Assistant Professor in the Department of Computer Engineering, Cairo University, Egypt. His research interests include parallel architectures, multicore systems, digital VLSI design, wireless sensor networks, and Networks-on-Chip (NoC) modeling, optimization, and performance evaluation.

**M. Watheq El-Kharashi** M. Watheq El-Kharashi received the Ph.D. degree in computer engineering from the University of Victoria, Victoria, BC, Canada, in 2002, and the B.Sc. degree (first class honors) and the M.Sc. degree in computer engineering from Ain Shams University, Cairo, Egypt, in 1992 and 1996, respectively. He is a Professor in the Department of Computer and Systems Engineering, Ain Shams University, Cairo, Egypt and an Adjunct Professor in the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada. His general research interests are in advanced system architectures, especially Networks-on-Chip (NoC), Systems-on-Chip (SoC), and secure hardware. He published about 100 papers in refereed international journals and conferences and authored two books and 6 book chapters.