Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 216406, 27 pages
doi:10.1155/2012/216406

Research Article

On Asymptotic Analysis of Packet and Wormhole Switched Routing Algorithm for Application-Specific Networks-on-Chip

Nitin

Department of CSE and ICT, Jaypee University of Information Technology, Waknaghat, Solan 173234, Himachal Pradesh, India

Correspondence should be addressed to Nitin, delnitin@ieee.org

Received 23 February 2012; Revised 11 June 2012; Accepted 13 June 2012

Academic Editor: Vivek Kumar Sehgal

The application of the multistage interconnection networks (MINs) in systems-on-chip (SoC) and networks-on-chip (NoC) is hottest since year 2002. Nevertheless, nobody used them practically for parallel communication. However, to overcome all the previous problems, a new method is proposed that uses MIN to provide intra-(global) communication among application-specific NoCs in networks-in-package (NiP). For this, four $O(n)^2$ fault-tolerant parallel algorithms are proposed. It allows different NoCs to communicate in parallel using either fault-tolerant irregular Penta multistage interconnection network (PNN) or fault-tolerant regular Hexa multistage interconnection network (HXN). These two are acting as an interconnects-on-chip (IoC) in NiP. Both IoC use packet switching and wormhole switching to route packets from source NoC to destination NoC. The results are compared in terms of packet losses and wormhole switching which comes out to be better than packet switching. The comparison of IoC on cost and MTTR concluded that the HXN has the higher cost than the PNN, but MTTR values of the HXN are low in comparison to the PNN. This signifies that the ability to tolerate faults and online repairing of the HXN is higher and faster than the PNN.

1. Introduction and Motivation

Parallel Processing refers to the concept of speeding-up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across n processors might execute n times faster than it would use a single processor.

It is known that one way for processors to communicate data is to use a shared memory and shared variables. However, this is unrealistic for large numbers of processors. A more realistic assumption is that each processor has its own private memory and data communication takeing place using message passing via an Interconnection Networks (INs).

INs originated from the design of high-performance parallel computers. INs make a major factor to differentiate modern multiprocessor architectures and are categorized according to a number of criteria such as topology, routing strategy, and switching technique. IN is building up of switching elements; topology is the pattern in which the individual switches are connected to other elements, like processors, memories, and other switches.

1.1. Interconnection Networks.

> "*Interconnection Networks should be designed to transfer the maximum amount of information within the least amount of time (and cost, power constraints) so as not to bottleneck the system.*"

INs have a long development history [1–8]. The Circuit switched networks have been used in telephony. In 1950s, the interconnection of computers and cellular automata as few prototypes was developed until 1960 when it awaited full use. Solomon in 1962 developed multicomputer network. Staran with its flip network, C.mmp with a crossbar and Illiac-IV with a wider 2D network received attention in early 1970s. This period also saw several indirect network used in vector and array processors to connect multiple processors to multiple memory banks. This problem was developed in several variants of MINs. The BBN Butterfly in 1982 was one of the first multiprocessors to use as an indirect network. The binary e-cube or hypercube network was proposed in 1978 and implemented in the Caltech Cosmic Cube in 1981. In the early 1980s, the academic focus was on mathematical properties of these networks and

became increasingly separated from the practical problems of interconnecting real systems.

The last decade was the golden period for INs research driven by the demanding communication problems of multicomputer enabled by the ability to construct single-chip Very Large Scale Integration (VLSI) routers, the researchers have made a series of breakthroughs that have revolutionized in digital communication systems. The Torus Routing Chip, in 1985, was one unique achievement. The first of a series of single-chip routing components introduced wormhole routing and virtual channels used for deadlock avoidance. The whole family of chip laid the framework for analysis of routing, flow-control, deadlock, and livelock issues in modern direct networks. A flurry of research followed with new theories of deadlock and livelock, new adaptive routing algorithms, and new methods for performance analysis. The research progressed in collective communication and network architectures on a regular basis. By the early 1990s, low-dimensional direct networks had largely replaced the indirect networks of the 1970s, and the hypercubes of the 1980s could be found in machines from Cray, Intel, Mercury, and some others. The applicability of INs in digital communication systems with the appearance of Myrinet was adopted in 1995. The point-to-point multiple networks technology replaced the use of buses, which were running into a limited performance due to electrical limits and were used in the barrier network in the Cray T3E, as an economical alternative to dedicated wiring. However, the interconnection network technology had certain barriers on design, and the various researchers and engineers have observed analysis of these networks [1, 4–8].

1.1.1. Multistage Interconnection Networks. As the acceptance and subsequent use of multiprocessor systems increased, the reliability, availability, performability, and performance characteristics of the networks that interconnect processors to processors, processors to memories, and memories to memories are receiving increased attention. A brief survey of INs and a survey of the fault-tolerant attributes of MINs are reported in [1–8]. A MIN in particular is an IN that consists of cascade of switching stages, contains switching elements (SEs). MINs are widely used for broadband switching technology and for multiprocessor systems. Besides this, MINs offer an enthusiastic way of implementing switches used in data communication networks. With the performance requirement of the switches exceeding several terabits/sec and teraflops/sec, it becomes imperative to make them dynamic and fault tolerant [9–14].

The typical modern day application of the MINs includes fault-tolerant packet switches, designing multicast, broadcast router fabrics, while SoCs and NoCs are hottest research topics in current trends [9–14]. Normally the following aspects are always considered while deigning the fault-tolerant MINs: the topology chosen, the routing algorithm used, and the flow control mechanism adhered. The topology helps in selecting the characteristics of the present chip technology in order to get the higher bandwidth, throughput, processing power, processor utilization, and probability of acceptance from the MIN-based applications, at an optimum hardware cost. Therefore, it has been decided to work on both irregular and regular fault-tolerant MINs as an application for NoCs.

1.2. Networks-in-Package. Networks-in-package (NiP) designs provide integrated solutions to challenging design problems in the field of multimedia and real-time embedded applications. The main characteristics of NiP platforms are as follows:

(1) networking between chip-to-chip in a single package,

(2) low development cost than NoC approach,

(3) low power consumption,

(4) high performance,

(5) small area.

Along with these characteristics, there are various fields to explore in NiP, which include the following:

(1) communication versus computation,

(2) deep Sub-Micron effect,

(3) power,

(4) global synchronization,

(5) heterogeneity of functions.

This paper focuses on an emerging paradigm that effectively addresses and presumably overcomes the many on-chip interconnection and communication challenges that already exist in today's chips or will likely occur in future chips. This new paradigm is commonly known as the NoC paradigm [15–18]. The NoC paradigm is one, if not the only one, fit for the integration of an exceedingly large number of computational, logic, and storage blocks in a single chip. Notwithstanding this school of thought, the adoption and deployment of NoC face important issues relating to design and test methodologies and automation tools. In many cases, these issues remain unresolved.

1.3. Networks-on-Chip. NoC is an emerging paradigm for communications within VLSI systems implemented on a single silicon chip. In a NoC system, modules such as processor cores, memories, and specialized Intellectual Property (IP) blocks exchange data using a network as a "public transportation" subsystem for the information traffic. A NoC is constructed from multiple point-to-point data links interconnected by switches, such that messages can be relayed from any source module to any destination module over several links, by making routing decisions at the switches. A NoC is similar to a modern telecommunications network, using digital bit-packet switching over multiplexed links. Although packet switching is sometimes claimed as a necessity for NoC, there are several NoC proposals utilizing circuit-switching techniques. This definition based on routers is usually interpreted so that a single shared bus, a single crossbar switch, or a point-to-point network is not NoC, but practically all other topologies are. This is somewhat confusing since all above mentioned are networks

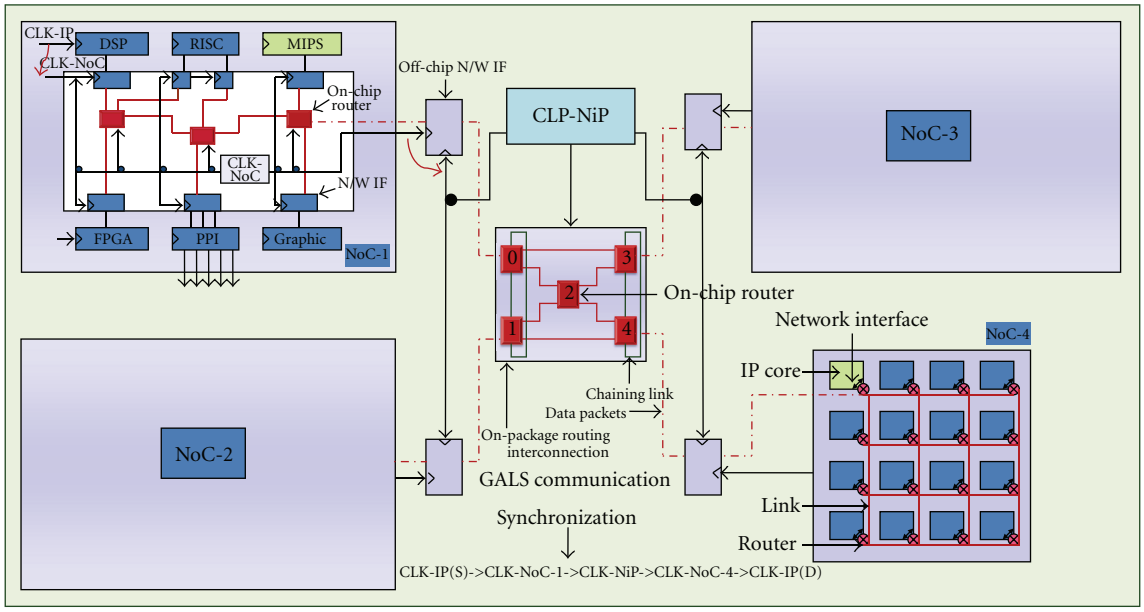
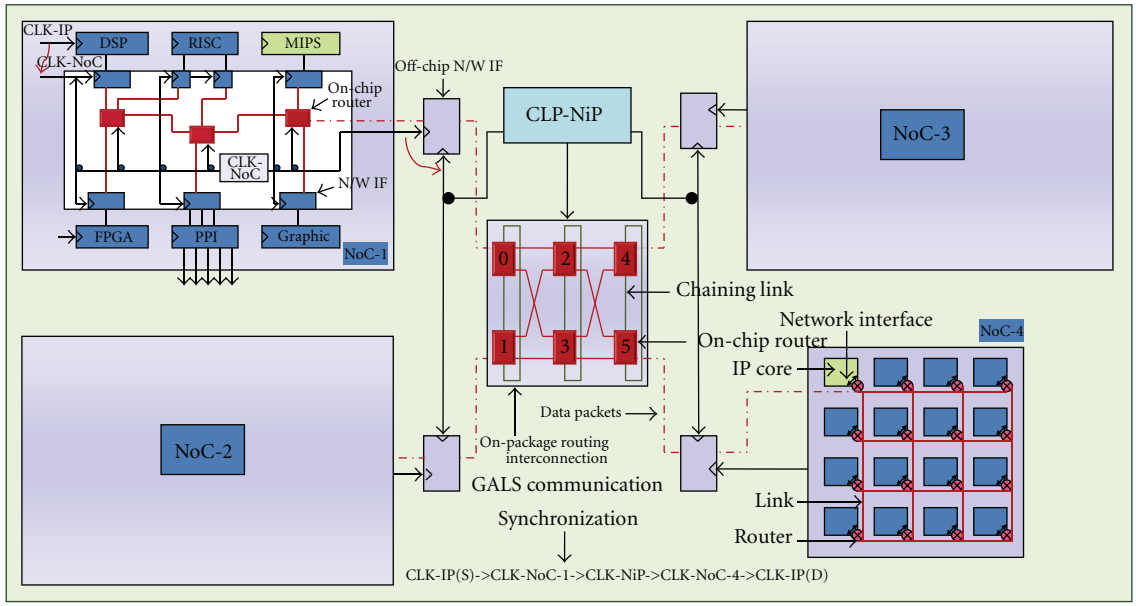
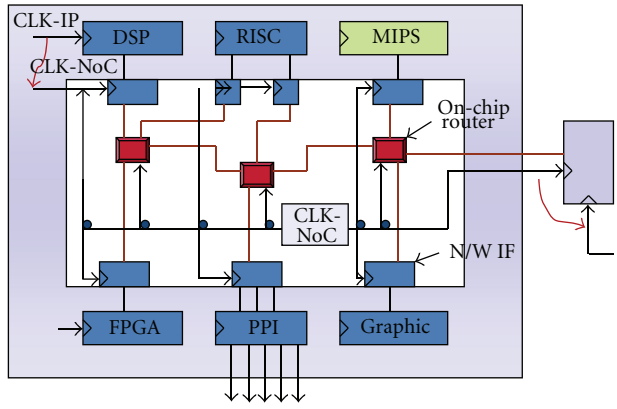


Figure 1: Parallel communication among various NoCs using PNN.





(a) Internal architecture of NoC-1 used in NiP.

Figure 2: Parallel communication among various NoCs using HXN.

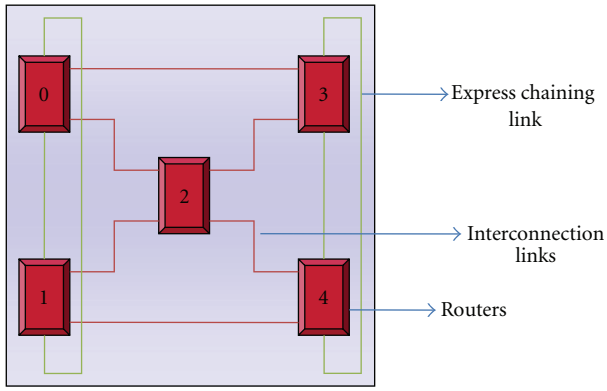


Figure 3: PNN as Interconnect-on-Chip.

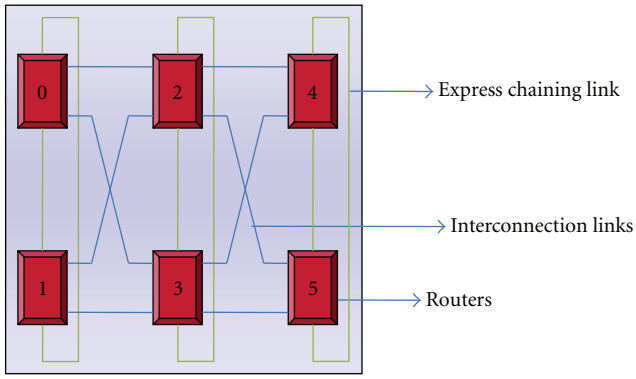


Figure 4: HXN as Interconnect-on-chip.

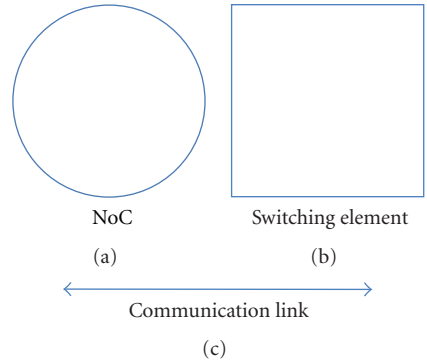


Figure 5: Following are the legends used in the simulation technique.

but are not considered as NoC. Note that some articles erroneously use NoC as a synonym for mesh topology although NoC paradigm does not dictate the topology. Likewise, the regularity of topology is sometimes considered as a requirement, which is obviously not the case in research concentrating on "Application-Specific NoC".

The wires in the links of the NoC are shared by many signals. A high level of parallelism is achieved, because all links in the NoC can operate simultaneously on different data packets. Therefore, as the complexity of integrated systems keeps growing, a NoC provides enhanced performance and scalability in comparison with previous communication architectures. Of course, the algorithms must be designed in such a way that it offers large parallelism and can hence utilize the potential of NoC.

Several forces drive the adoption of NoC architecture: from a physical design viewpoint, in nanometer Complementary Metal-Oxide Semiconductor (CMOS) technology interconnects dominate both performance and dynamic power dissipation, as signal propagation in wires across the chip requires multiple clock cycles. NoC links can reduce the complexity of designing wires for predictable speed, power, noise, reliability, and so on, thanks to their regular, well-controlled structure. From a system design viewpoint, with the advent of multicore processor systems, a network is a natural architectural choice. A NoC can provide separation between computation and communication, support modularity and IP reuse via standard interfaces, handle synchronization issues, serve as a platform for system test, and, hence, increase engineering productivity.

Although NoC can borrow concepts and techniques from the well-established domain of computer networking, it is impractical to reuse features of old networks and symmetric multiprocessors. In particular, NoC switches should be small, energy efficient, and fast. Neglecting these aspects along with proper, quantitative comparison was typical for early NoC research, but today all are considered in more detail. The routing algorithms must be implemented by simple logic, and the number of data buffers should be minimal. Network topology and properties may be Application Specific. NoC need to support quality of service, namely, achieve the various requirements in terms of throughput, end-to-end delays, and deadlines. To date, several prototypes of NoCs are designed and analyzed in industry and academia. However, only few are implemented on silicon. However, many challenging research problems remain to be solved at all levels, from the physical link level through the network level and all the way up to the system architecture and application software.

Most NoC are used in embedded systems, which interact with their environment under more or less hard time constraints. The communication in such systems has a strong influence on the global timing behavior. Methods are needed to analyze the timing, as average throughput as well as worst-case response time [17]. However, from a VLSI design perspective, the energy dissipation profile of the interconnect architectures is of prime importance as the latter can represent a significant portion of the overall energy budget. The silicon area overhead due to the interconnect fabric is important too. The common characteristic of these kinds of architectures is such that the processor/storage cores communicate with each other through high-performance links and intelligent switches and such that the communication design is represented at a high abstraction level. The different NoC topologies are already used in [19], and these topologies give different communication structure in NoC [20].

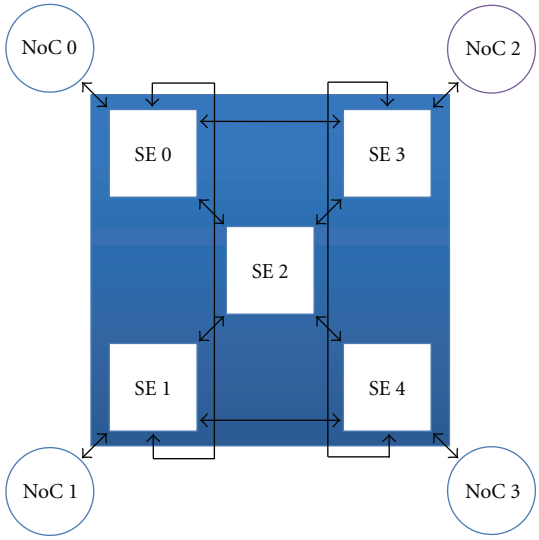


Figure 6: The initial architectural NiP model using PNN.

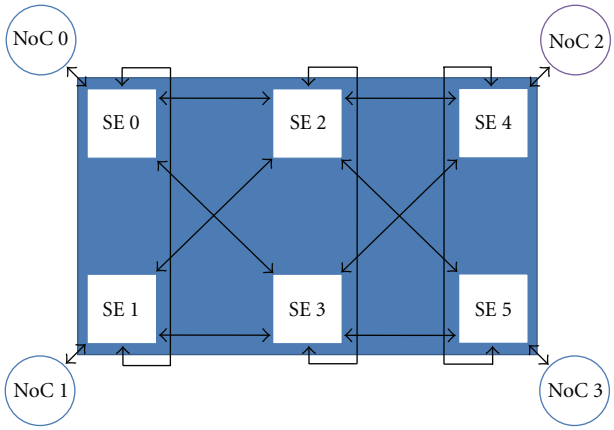


Figure 7: The initial architectural NiP model using HXN.

The application of the MIN in SoCs [10, 15–18] and NoCs [10, 16–18] is consistently drawing attention since year 2002. The parallel communication among Application-Specific NoC [9, 10] is a major problem to handle by the researchers. Nevertheless, nobody used them practically for parallel communication. The literature survey reveals that Star, Common Bus, and Ring topologies were used as a medium to set up intra-NoCs communication [21]. However, these communication systems have many tradeoffs and disadvantages as mentioned below:

(1) high latency,

(2) low scalability,

(3) poor performance,

(4) zero fault-tolerance,

(5) no On-chip repairability,

(6) high contention over sharing of channel,

(7) presence of livelock,

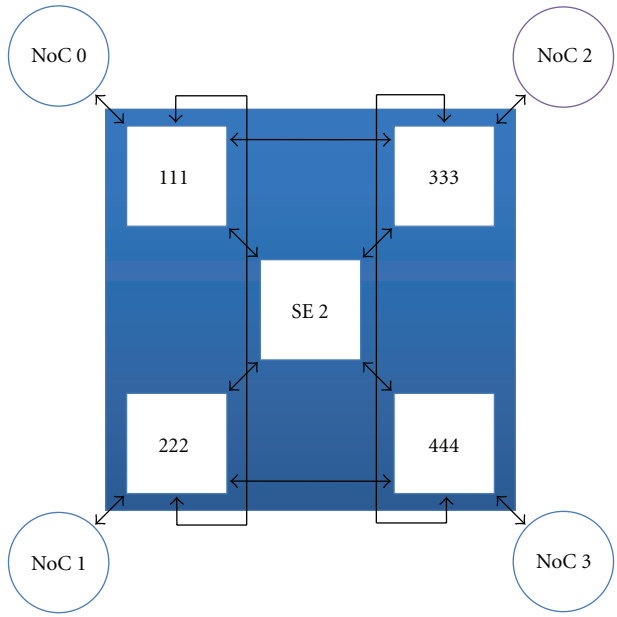


Figure 8: The simulation state of NiP-PNN model at first step of best case.

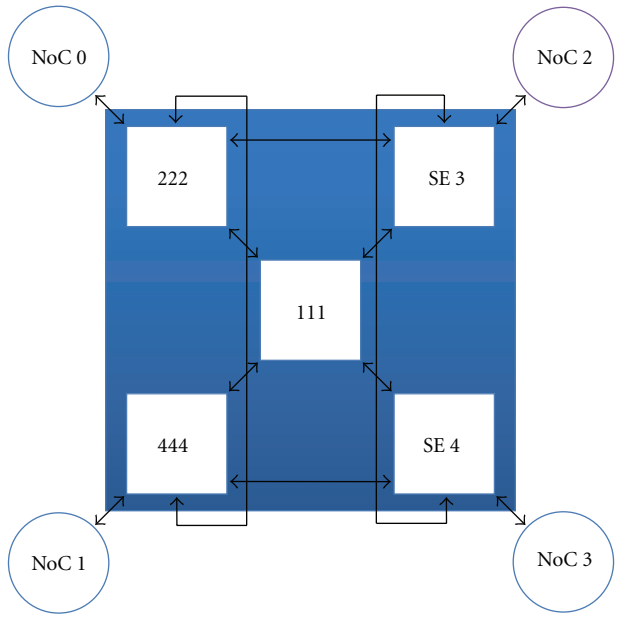


Figure 9: The simulation state of NiP-PNN model at second step of best case.

(8) presence of deadlock,

(9) low probability of acceptance of data packets.

However, to overcome all the previous problems, for the first time this paper proposes a new method that sets up intra-(global) communication between Application-Specific (heterogeneous or homogenous) NoCs in NiP. The said architecture uses $O(n)^2$ time fault-tolerant packet and worm-hole switching parallel algorithms. These algorithms allow different NoCs to communicate efficiently in parallel with

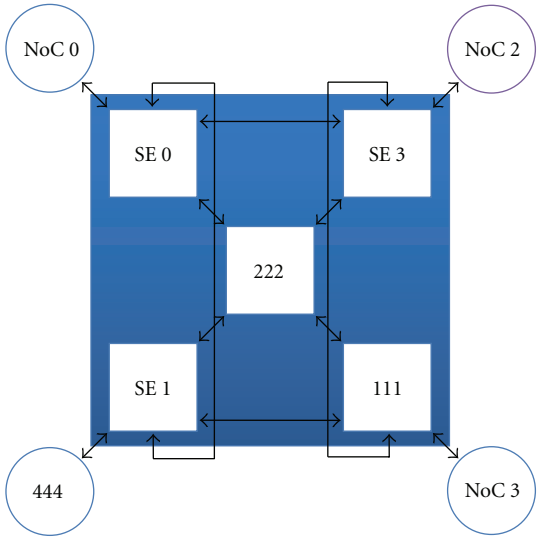


Figure 10: The simulation state of NiP-PNN model at third step of best case.

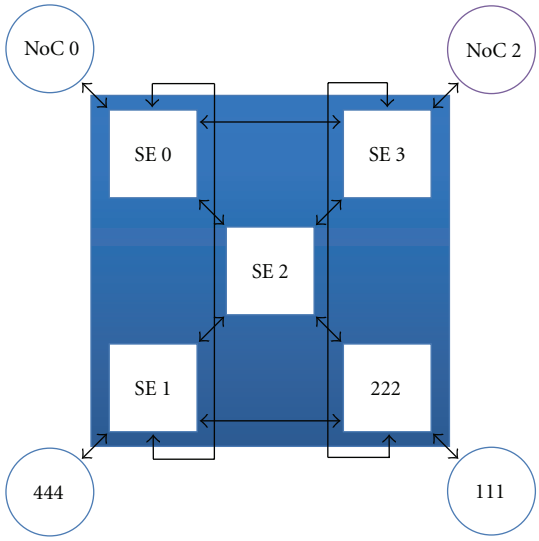


Figure 11: The simulation state of NiP-PNN model at fourth step of best case.

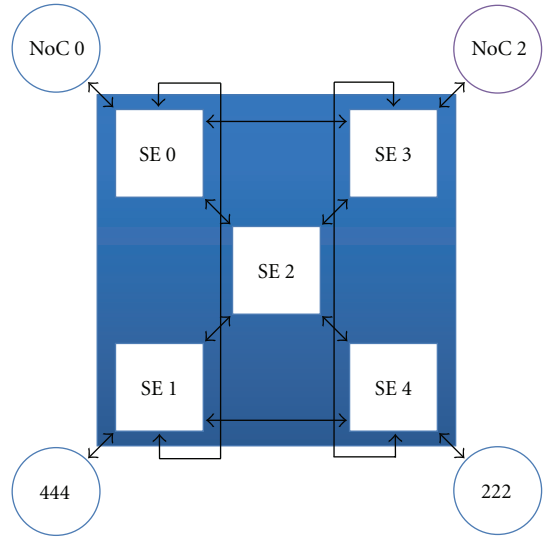


Figure 12: The simulation state of NiP-PNN model at fifth step of best case.

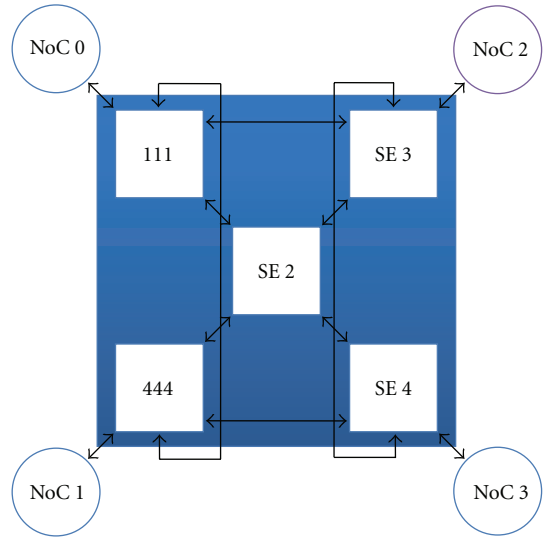


Figure 13: The simulation state of NiP-PNN model at first step of worst case.

minimum number of the packet losses. Out of the two IoCs, one is 2×2 fault-tolerant irregular Penta multistage interconnection networks (PNNs), with 3 stages and 5 SEs (all stages include chaining or express links, except the middle one) and other is 2×2 fault-tolerant regular Hexa multistage interconnection networks (HXNs), with 3 stages and 6 SEs (all stages include chaining or express links, except the middle one).

The rest of the paper is organized as follows: Section 2 describes the general NiP architecture including the fault-tolerant parallel algorithm designed to provide parallel communication among different NoCs using HXN and PNN followed by the their comparisons on Cost and Mean Time to Repair (MTTR). Section 3 provides the conclusion followed by the references.

2. Application-Specific NiP Architecture Using Irregular PNN and Regular HXN

The general architecture of NiP resembles with the Open Systems Interconnection (OSI) Model. The Physical layer refers to all that concerns the electric details of wires, the circuits and techniques to drive information, while the Data Link level ensures a reliable transfer regardless of any unreliability in the physical layer and deals with medium access. At the Network level there are issues related to the topology and the consequent routing scheme, while the Transport layer manages the end-to-end services and the packet segmentation/reassembly. Upper levels can be viewed merged up to the Application as a sort of adaptation layer that implements services in hardware or through part of

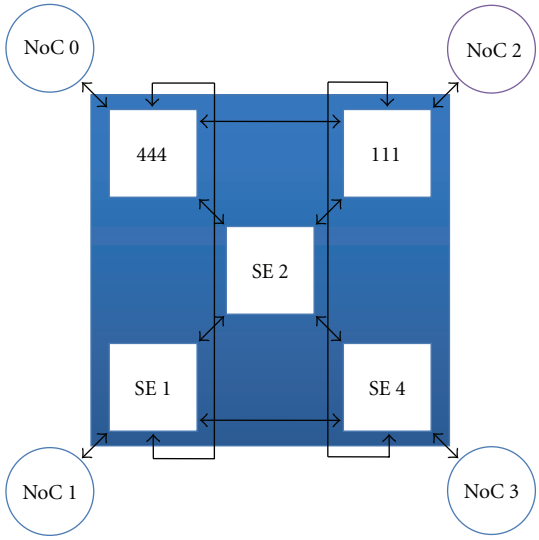


Figure 14: The simulation state of NiP-PNN model at second step of worst case.

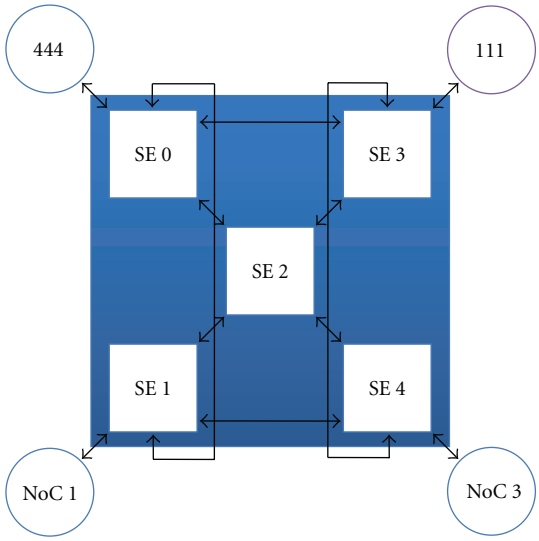


Figure 15: The simulation state of NiP-PNN model at third step of worst case.

an operating systems and exposes the NoC infrastructure according to a proper programming model, for example, the Message Passing (MP) paradigm.

NiP is a specific approach, which provides the common interface through which all NoC can communicate together more efficiently and robustly. It contains three different types of building blocks, appropriately interconnected to each other and a patented network topology that promises to deliver the best price/performance trade-off in future NiP applications as follows [20].

Figures 1 and 2 show a general NiP architecture in which four NoC chips are mounted on a single package. These NoC communicate with each other through an intermediate chip, known as IoC [22].

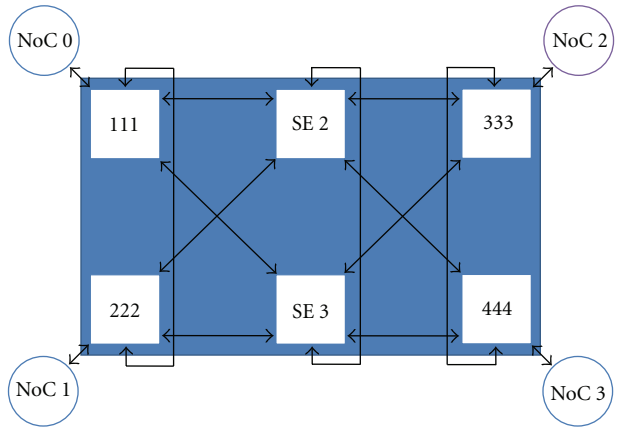


Figure 16: The simulation state of NiP-HXN model at first step of best case.

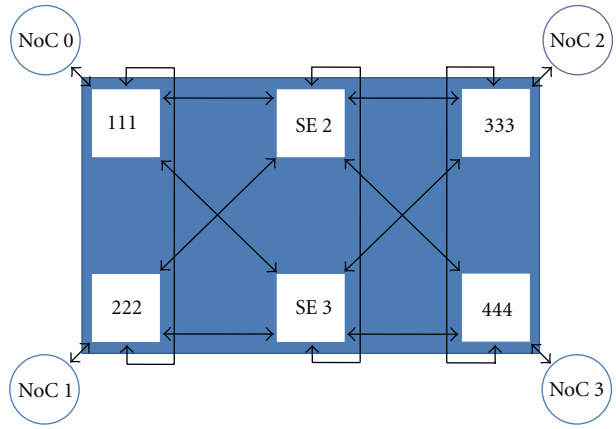


Figure 17: The simulation state of NiP-HXN model at second step of best case.

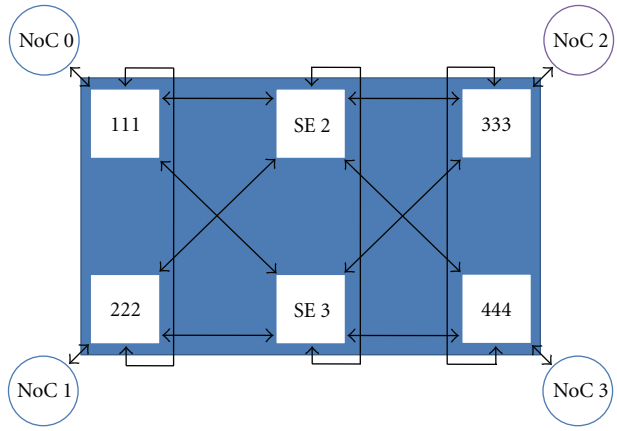


Figure 18: The simulation state of NiP-HXN model at third step of best case.

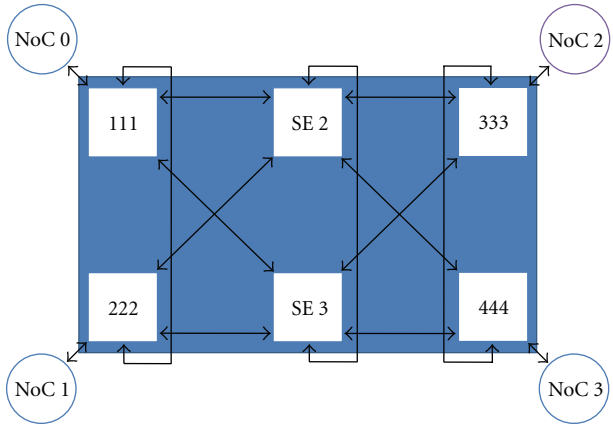


Figure 19: The simulation state of NiP-HXN model at fourth step of best case.

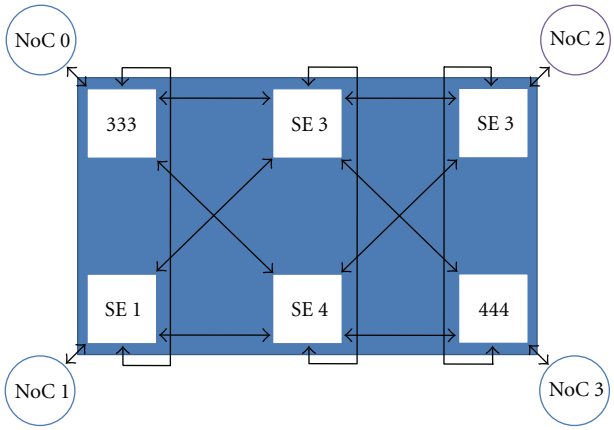


Figure 20: The simulation state of NiP-HXN model at first step of worst case.

2.1. Interconnects-on-Chip Architecture. Figures 3 and 4 show the different types of architecture of IoC, that is, one belongs to the class of irregular fault-tolerant MIN and other belongs to the class of regular fault-tolerant MIN. This first chip, shown in Figure 3, consists of five routers; working as Switching Elements (SEs) is known as PNN and Figure 4 shows the architecture of HXN with 6 SEs.

These routers are connected with the main link and chaining or express links, which makes the IoC highly, fault tolerant. Here the architectural design of IoC is similar to a small MIN, widely used for broadband switching technology and for multiprocessor systems. Besides this, it offers an enthusiastic way of implementing switches/routers used in data communication networks. With the performance requirement of the switches/routers exceeding several terabits/sec and teraflops/sec, it becomes imperative to make them dynamic and fault tolerant. The typical modern day application of the MIN includes fault-tolerant packet switches, designing multicast, broadcast router fabrics while SoC and NoC are hottest now days [9–14].

Table 1: Initial source and destination information used to set up communication between NoCs.

Source NoC	Destination NoC	Payload
0	3	111
1	3	222
2	3	333
3	1	444

Table 2: Intermediate SE information while data moves from source NoC to destination NoC at first step of best case.

Source NoC	Destination NoC	Switching element at first step
0	3	0
1	3	1
2	3	3
3	1	4

Table 3: Intermediate SE information while data moves from source NoC to destination NoC at second step of best case.

Source NoC	Destination NoC	Status/switching element at second step
0	3	2
1	3	0
2	3	Destroyed
3	1	1

Table 4: Intermediate SE information while data moves from source NoC to destination NoC at third step of best case.

Source NoC	Destination NoC	Status/switching element at third step
0	3	4
1	3	2
3	1	Reached successfully at NoC 1

Table 5: Intermediate SE information while data moves from source NoC to destination NoC at fourth step of best case.

Source NoC	Destination NoC	Status/switching element at third step
0	3	Reached successfully at NoC 3
1	3	4
3	1	Reached successfully at NoC 1

2.2. Switching Methodologies, Testbed, and Assumptions. Switching techniques determine when and how internal switches connect their inputs to outputs and the time at which message components may be transferred along these paths. For uniformity, the same approach for all NoC architectures has been used here. There are different types of switching techniques [6–8] as follows.

Definition 1 (*Circuit Switching*). A physical path from source to destination is reserved prior to the transmission of the data. The path is held until all the data has been

```
Input: Number of nodes.
Output: Pairs
(1) BEGIN
(2)    pair()
(3)     Open one.doc, two.doc, three.doc, four.doc file to write output;
(4)     Input four nodes from user;
(5)     k, c, d, e, l, g = 0;
(6)    FOR i = 0 to 3
(7)       FOR j = 0 to 3
(8)           arr[k][0] = node[i];
(9)           arr[k][1] = node[j];
(10)         k++;
(11)      END FOR
(12)   END FOR
(13)   FOR i = 0 to 12
(14)    Print (arr[i][0], arr[i][1]) to one.doc
(15)       FOR j = 0 to 12
(16)         IF (i ! = j), THEN
(17)           Print 2-2 pairs (arr2[i][0], arr2[i][1], arr2[j][0], arr2[j][1]) to two.doc;
(18)           c++;
(19)        FOR d = 0 to 12
(20)           IF (j ! = d and i ! = d), THEN
(21)           Print 3-3 pairs (arr2[i][0], arr2[i][1], arr2[j][0], arr2[j][1], arr2[d][0], arr2[d][1]) to three.doc;
(22)           e++;
(23)          FOR l = 0 to 12
(24)              IF (l ! = j && l ! = d && l ! = i), THEN
(25)              Print 4-4 pairs
(26)              (arr2[i][0], arr2[i][1], arr2[j][0], arr2[j][1], arr2[d][0], arr2[d][1], arr2[l][0], arr2[l][1]) to four.doc;
(27)              g++;
(28)             END FOR
(29)           END FOR
(30)         END FOR
(31)   END FOR
(32)    Print number of 1-1 pairs = k to one.doc;
(33)    Print number of 2-2 pairs = c to two.doc;
(34)    Print number of 3-3 pairs = e to three.doc;
(35)    Print number of 4-4 pairs = g to four.doc;
(36)   END pair()
(37) END

The run time complexity of the Algorithm: Test cases is O(n)^4
```

Algorithm 1: Test cases.

Table 6: The payload successfully reached to its desired destination NoC.

Source NoC	Destination NoC	Status
0	3	Reached successfully at NoC 3
1	3	Reached successfully at NoC 3
3	1	Reached successfully at NoC 1

Table 7: Initial source and destination information used to set up communication between NoCs.

Source NoC	Destination NoC	Payload
0	2	111
1	3	222
1	2	333
1	0	444

transmitted. The advantage of this approach is that the network bandwidth is reserved for the entire duration of data. However, valuable resources are also tied up for the duration of the transmitted data, and the set up of an end-to-end path causes unnecessary delays [5–8].

Definition 2 (*Packet Switching*). A data is divided into fixed-length blocks called packets and instead of establishing a path before sending any data, whenever the source has a packet to be sent, it transmits the data. The need for storing entire packets in a switch in case of conventional packet switching makes the buffer requirement high in these cases. In a SoC environment, the requirement is that switches should not consume a large fraction of silicon area compared to the IP blocks [5–8].

```
Input: n, stores Number of parallel processing NoC.
Source, a NoC type array, stores the Source NoC Numbers a part of NoC structure.
Destination, a NoC type array, stores the Destination NoC Numbers a part of NoC structure.
Payload, a part of NoC structure that holds the data generated as Source NoC.
Output: Payload.

(1)  BEGIN
(2)    FOR (I = 0 to 4) DO /*Initialization of Elements of Switching Element structure*/
(3)     Info = NULL
(4)     Info1 = NULL
(5)     Source = −999
(6)     Destination = −999
(7)    END FOR
(8)    FOR (I = 0 to 3) DO
(9)     Payload = 0 /*Initializing payload of all the 4 NoC to 0*/
(10)   Number = I /*Numbering of all the 4 NoC from 0 to 3*/
(11)  END FOR
(12)     Get the Number of parallel communicating NoC, n
(13)  FOR (l = 0 to n) DO
(14)     Get the Source and Destination NoC S[l], D[l]
(15)     Get the respective payloads
(16)  END FOR
(17)  FOR (x = 0 to n) DO
(18)   BEGIN

/*Stage1: Transferring the "Payload" values from the NoC to the respective "Info" values of Switching Element structure*/

(19)     FOR (y = 0 to n−1) DO
(20)     Transfer the payloads of NoC to the Info of next immediate Switching Elements
(21)      Transfer the Source NoC Number to the Source of Switching Element
(22)      Transfer the Destination Number to the Switching Element
(23)    END FOR

/*Stage2: If the communicating NoC are on the same side then check the source and respective destination number of the
Switching Element and transferring it to the next Switching Element having empty "Info" value*/

(24)    FOR (t = 0 to 2) DO
(25)      Check Source and respective Destination pair DO
(26)       Check Info of first linked Switching Element DO
(27)         IF Info = NULL
(28)           Transfer Info into this Switching Element
(29)           Transfer the Source and Destination Number to this Switching Element
(30)         END IF

/*If "Info" of first linked Switching Element is not empty then transfer the packet to the "Info" of Second linked Switching
Element*/

(31)             ELSE
(32)               Check Info of Second linked Switching Element DO
(33)                 IF Info = NULL
(34)                   Transfer Info into Switching Element
(35)                   Transfer the Source and Destination Number to the Switching Element
(36)                 END IF

/*Stage2: If communicating NoC are on the opposite side then Check the source and respective destination number of the
Switching Element and transferring it to Switching Element having empty "Info" value*/

(37)         Check Source and respective Destination pair DO
(38)         Check Info of first linked Switching Element DO
(39)           IF Info = NULL
(40)             Transfer Info into this Switching Element
(41)             Transfer the Source and Destination Number to this Switching Element
(42)           END IF

/*If the "Info" of first linked Switching Element is not empty then transfer the packet to "Info" of Second linked Switching
Element*/
```

Algorithm 2: Continued.

```
(43)          ELSE
(44)            Check Info of Second linked Switching Element DO
(45)              IF Info = NULL
(46)                Transfer Info into Switching Element
(47)                  Transfer the Source and Destination Number to the Switching Element
(48)              END IF

/*If the "Info" of first and Second linked Switching Elements are not empty then transfer "Info" To third linked Switching
Element*/

(49)          ELSE
(50)            Check Info of third linked Switching Element DO
(51)              IF Info = NULL
(52)                Transfer Info into Switching Element
(53)                  Transfer the Source and Destination Number to the Switching Element
(54)              END IF
/*Stage3: If the linked NoC is the destination that is, NoC-0 for SE[0], NoC-1 for SE[1] NoC-3 for SE[2], NoC-4 for SE[3]
then transfer the "Info" of the Switching Element to their respective linked NoC*/
(55)      IF Destination of SE[0] = 0
(56)        Transfer Info to payload of NoC-0

/*If the linked NoC is not the destination then transfer the "Info" of Switching Element (SE) to the next empty
Switching Element. If not then destroy the packet*/

(57)        ELSE
(58)          Transfer Info to first linked Switching Element
(59)      END IF
(60)      IF Destination of SE[1] = 1
(61)        Transfer Info to payload of NoC-1
(62)          ELSE
(63)            Transfer Info to first linked Switching Element
(64)      END IF
(65)      IF Destination of SE[3] = 2
(66)        Transfer Info to payload of NoC-2
(67)          ELSE
(68)            Transfer Info to first linked Switching Element
(69)      END IF
(70)      IF Destination of SE[4] = 3
(71)        Transfer Info to payload of NoC-3
(72)          ELSE
(73)            Transfer Info to first linked Switching Element
(74)      END IF

/*Stage4: Transfer the Info of the Switching Element (that could not be transferred in the previous stage) to the destination
NoC*/

(75)      Transfer Info of Switching Element to the payload of the Destination NoC

(76)    END FOR
(77)  END BEGIN
(78) END FOR
(79) END BEGIN
```

The run time Complexity of the Algorithm: NoC_PS_IRREGULAR_PNN is $O(n)^2$

Algorithm 2: NoC_PS_IRREGULAR_PNN.

Definition 3 (*Wormhole Switching*). The packets are divided into fixed length flow control units (flits), and the input and output buffers are expected to store only a few flits. As a result, the buffer space requirement in the switches can be small compared to that generally required for packet switching. Thus, using a wormhole switching technique, the switches will be small and compact. The first flit, that is, header flit, of a packet contains routing information. Header flit decoding enables the switches to establish the path and subsequent flits simply follow this path in a pipelined fashion. As a result, each incoming data flit of a message packet is simply forwarded along the same output

```
Input: n, stores Number of parallel processing NoC.
Source, a NoC type array, stores the Source NoC Numbers a part of NoC structure.
Destination, a NoC type array, stores the Destination NoC Numbers a part of NoC structure.
Payload, a part of NoC structure that holds the data generated as Source NoC.
Output: Payload.

(1)  Initialize global variable
      Set graph[5, 5] = [(0, 1, 1, 1, 0), (1, 0, 1, 0, 1), (1, 1, 0, 1, 1), (1, 0, 1, 0, 1), (0, 1, 1, 1, 0)]
      busy[5] = [0, 0 · · · 0]
      success[4] = [0, 0, 0, 0]
(2)  Form a class packet
       Attributes:
           PUBLIC header[2]: INTEGER
           PUBLIC flit[4]: INTEGERs:
        METHOD: PUBLIC packet()
          BEGIN
              flit[0] = 0
              flit[1] = 0
              flit[2] = 0
              flit[3] = 0
          End
        End METHOD
      End class packet
      packet p[4]
(3)  Open the file "pairs.txt" with handle file1
      Open the file "pairs1.txt" with handle file2
      FOR pairs = 3 to pairs = 4
        WHILE (EOF of file1|file2 is not reached)
          IF (pairs = 3)
              getline from file1 = ar
          ELSE
              getline from file2 = ar
              [call puts with ar] puts (ar)
      FOR i = 0 to 2
       Input source and destination of NoC packet
       [Initialize s and d] Set = ar[i ∗ 4] − 48 and d = ar[i ∗ 4 + 2] − 48
(4)  IF s <= 2 THEN
         [Initialize header] header[0] = s−1
          ELSE
             header[0] = s
          IF d <= 2 THEN
             [Initialize header] header[1] = d−1
          ELSE
             [Initialize header 1] header[1] = d
      End FOR
(5)  Output header for all packets
       Initialize flit and busy FOR i = 0 to 3
       p[i].flit = p[i].header[0] and busy[i].header[i] = 1
(6)  FOR k = 0 to [value of pairs]
       IF flit = header[1] and flit ! = 99 and flit ! = 100 THEN
        flit = 99
(7)  IF graph [flit][header] = = 1 and flit ! = 99 and flit ! = 100 THEN
       IF busy = = 1 THEN
       IF busy[2] = = 0 THEN
        Assign flit[0] = 0
        Assign busy[2] = 1
        Assign flit = 2 [0]
(8)  IF busy[3] = 0
       Initialize flit[0] = 0
       Assign busy[3] = 1
       Assign flit[0] = 3
(9)  ELSE DO
       Assign flit[0] = 0
       Assign flit[0] t = header[1]
```

Algorithm 3: Continued.

```
        Assign flit[0] = 1
(10) ELSE IF flit[0] ! = 9 and flit[0] ! = 100 THEN
        Initialize flag = 0
         FOR x = 0 to 5
          Set flag = 0
        FOR x = 0 to 5
          IF flit(x) = 1 and busy(x) = 0
           Assign flit[0] = 0
           Assign flit[0] = x
             Assign flit[0] = 1
             Assign flag = 1
(11) IF flag = 0
        Assign flit[0] = 100
(12) Print flit[k] FOR k = 0 to [value of pairs]
(13) Assign busy[p[k].flit[3]] = 0
        Assign flit[3] = flit[2]]
        Assign busy[p[k].flit[3]] = 1
        Assign flit[2] = flit[1]]
        Assign busy[p[k].flit[3]] = 1
        Assign flit[1] = flit[0]]
        Assign busy[p[k].flit[3]] = 1
(14) Loop e = 1 to pairs
        IF flit(3) = 99
          Increment success[pairs − 1]
(15) Loop z = 1 to z = pairs
          Assign success[z] = success[z]/z
          Print [success[z] and efficiency]
          Efficiency = (success[z]/total[z])∗100

The run time complexity of the Algorithm: NoC_WS_IRREGULAR_PNN is O(n)^2
```

Algorithm 3: NoC_WS_IRREGULAR_PNN.

Table 8: Intermediate SE information while data moves from source NoC to destination NoC at first step of worst case.

Source NoC	Destination NoC	Status/switching element at first step
0	2	0
1	3	Destroyed
1	2	Destroyed
1	0	444

Table 9: Intermediate SE information while data moves from source NoC to destination NoC at second step of worst case.

Source NoC	Destination NoC	Switching element at second step
0	2	3
1	0	0

Table 10: The payload successfully reached to its desired destination NoC.

Source NoC	Destination NoC	Status
0	2	Reached successfully at NoC 2
1	0	Reached successfully at NoC 0

channel as the preceding data flit, and no packet reordering is required at destinations. If a certain flit faces a busy channel, subsequent flits also have to wait at their current locations [5–8].

We have used packet and wormhole switching algorithms to send data from one NoC to other NoC in parallel environment. However, wormhole routing is better in today's scenario but considering the low cost, as it is a measure factor for system performance. The inclusion of buffer will complex the current system and the cost would increase smartly.

Deadlocks, livelocks, and starvation arise because the number of resources is finite. Additionally, some of these situations may produce the others. For instance, a deadlock permanently blocks some packets. As those packets are occupying some buffers, other packets may require them to reach their destination, being continuously misrouted around their destination node and producing livelock. It is extremely important to remove deadlocks, livelocks, and starvation when implementing an interconnection network. Otherwise, some packets may never reach their destination. The following definitions and the problems are very important therefore, care should be taken while designing the code.

Definition 4 (*Deadlock*). A deadlock occurs when some packets cannot advance toward their destination because the buffers requested by them are full. A packet may be permanently blocked in the network because the destination node does not consume it. This kind of deadlock is produced by the application [5–8].

```
Input: n, stores Number of parallel processing NoC.
Source, a NoC type array, stores the Source NoC Numbers a part of NoC structure.
Destination, a NoC type array, stores the Destination NoC Number a part of NoC structure.
Payload, a part of NoC structure that holds the data generated as Source NoC.
Output: Payload.

(1)   BEGIN

(2)    FOR (I = 0 to 6) DO /*Initialization of Elements of Switching Element structure*/
(3)     Info = NULL
(4)     Info1 = NULL
(5)     Source = −999
(6)     Destination = −999
(7)    END FOR

(8)    FOR (I = 0 to 4) DO
(9)     Payload = 0 /*Initializing payload of all the 4 NoC to 0*/
(10)    Number = I /*Numbering of all the 4 NoC from 0 to 3*/
(11)   END FOR
(12)  Get the Number of parallel communicating NoC, n
(13)  FOR (l = 0 to n) DO
(14)   Get the Source and Destination NoC S[l], D[l]
(15)   Get the respective payloads
(16)  END FOR
(17)  FOR (x = 0 to 6) DO
(18)    BEGIN

/*Stage1: Transferring the "Payload" values from the NoC to the respective "Info" values of Switching
Element structure*/

(19)      FOR (y = 0 to n−1) DO
(20)      Transfer the payloads of NoC to the Info of next immediate Switching Elements
(21)      Transfer the Source NoC Number to the Source of Switching Element
(22)      Transfer the Destination Number to the Switching Element
(23)      END FOR

/*Stage2: If the communicating NoC are on the same side then check the source and respective destination
number of the Switching Element and transferring it to the next Switching Element having empty "Info" value*/

(24)      FOR (t = 0 to 2) DO
(25)        Check Source and respective Destination pair DO
(26)        Check Info of chain linked Switching Element DO
(27)          IF Info = NULL
(28)            Transfer Info into this Switching Element
(29)            Transfer the Source and Destination Number to this Switching Element

/*If "Info" of first linked Switching Element is not empty then transfer the packet to the "Info" of second linked Switching
Element*/

(30)          ELSE
(31)            Check Info of first linked Switching Element DO
(32)              IF Info = NULL
(33)                Transfer Info into Switching Element
(34)                Transfer the Source and Destination Number to the Switching Element
(35)          ELSE
(36)            Check Info of second linked Switching Element DO
(37)              IF Info = NULL
(38)                Transfer Info into Switching Element
(39)                Transfer the Source and Destination Number to the Switching Element

/*Stage3: If communicating NoC are on the opposite side checking the source and respective destination number of the
Switching Element and transferring it to Switching Element having empty "Info" value*/
```

Algorithm 4: Continued.

```
(40)          Check Source and respective Destination pair DO
(41)          Check Info of first linked Switching Element DO
(42)          Transfer Info into this Switching Element
(42)                    IF Info = NULL
(44)           Transfer the Source and Destination Number to this Switching Element

/*If the "Info" of first linked Switching Element is not empty then transfer of packet to "Info" of second linked Switching
Element*/

(45)            ELSE
(46)            Check Info of second linked Switching Element DO
(47)            Transfer Info into Switching Element
(48)              IF Info = NULL
(49)                Transfer the Source and Destination Number to the Switching Element

/*If the "Info" of first and second linked Switching Elements are not empty then transfer "Info" to third linked Switching
Element*/

(50)            ELSE
(51)              Check Info of chain linked Switching Element DO
(52)              Transfer Info into Switching Element
(53)               IF Info = NULL
(54)                 Transfer the Source and Destination Number to the Switching Element

(55)              Check Source and respective Destination pair DO
(56)              Check Info of first linked Switching Element DO
(57)            IF Info = NULL DO
(58)              Transfer Info into this Switching Element
(59)              Transfer the Source and Destination Number to this Switching Element

/*If the "Info" of first linked Switching Element is not empty then transfer of packet to "Info" of second linked Switching
Element*/

(60)            ELSE
(61)              Check Info of second linked Switching Element DO
(62)            IF Info = NULL DO
(63)              Transfer Info into Switching Element
(64)              Transfer the Source and Destination Number to the Switching Element

/*If the "Info" of first and second linked Switching Elements are not empty then transfer "Info" to third linked Switching
Element*/

(65)                ELSE
(66)                  Check Info of chain linked Switching Element DO
(67)            IF Info = NULL DO
(68)              Transfer Info into Switching Element
(69)              Transfer the Source and Destination Number to the Switching Element

/*Stage4: If the linked NoC is the destination, that is, NoC-0 for SE[0], NoC-1 for SE[1] NoC-3 for SE[4], NoC-4 for SE[5]
then transfer the "Info" of the Switching Element to their respective linked NoC*/

(70)          IF Destination of SE[0] = 0 DO
(71)           Transfer Info to payload of NoC-0
(72)          ELSE
(73)            Transfer Info to first linked Switching Element
(74)          END IF
(75)          IF Destination of SE[1] = 1 DO
(76)           Transfer Info to payload of NoC-1
(77)            ELSE
(78)             Transfer Info to first linked Switching Element
(79)          END IF

(80)          IF Destination of SE[3] = 2 DO
(81)           Transfer Info to payload of NoC-2
(82)            ELSE
(83)              Transfer Info to first linked Switching Element
(84)           END IF
```

Algorithm 4: Continued.

```
(85)              IF Destination of SE[4] = 3
(86)                Transfer Info to payload of NoC-3
(87)                 ELSE
(88)                   Transfer Info to first linked Switching Element
(89)              END IF

/* Stage5: Transfer the Info of the Switching Element that could not be transferred in the previous stage) to the destination
NoC*/

(90)              Transfer Info of Switching Element to the payload of the destination NoC
(91)            END FOR
(92)          END BEGIN
(93)        END FOR
(94)      END BEGIN

The run time complexity of the algorithm: NoC_PS_REGULAR_HXN is O(n)^2.
```

Algorithm 4: NoC_PS_REGULAR_HXN.

Definition 5 (*Livelock*). A situation when some packets are not able to reach their destination, even if packets never block permanently. A packet may be traveling around its destination node, never reaching it because the channels required to do so are occupied by other packets. It can only occur when packets are allowed to follow nonminimal paths [5–8].

Definition 6 (*Starvation*). Packet may be permanently stopped if traffic is intense and the resources requested by it are always granted to other packets also requesting them. It usually occurs when an incorrect resource assignment scheme is used to arbitrate in case of conflict [5–8].

The two types of policies, which exist while transferring the packets through IoC to NoCs in parallel communication environment, are as follows.

Definition 7 (*Milk Policy*). This policy states that the newer packet kills the previous residing packet. The older packet is destroyed.

Definition 8 (*Wine Policy*). This policy states that the older packet will survive and the newer packet gets destroyed or in other words whenever a new transfer takes place the next SE (in respect to the current SE) is checked for ideal condition and if it is not ideal then no transfer takes place, that is, older packet resides and new arriving packet gets destroyed.

This paper uses wine policy as it is the best according to our problem.

2.2.1. Output Simulation Scenario to Simulate the NiP Architecture Using PNN as IoC. The following are assumptions to simulate the NiP architecture using PNN as IoC.

(1) No packet can survive for more than 5 clock cycles where each clock cycle equals to 1 for loop.

(2) During same side communication (0-1, 1-0, 2-3, 3-2) priority is given to the chaining link, after that straight path or exchange path accordingly.

(3) During straight communication (0–2, 2–0, 1–3, 3–1) priority is given to straight Path, then exchange and at last to the chain links.

(4) The communication in program takes place accordingly as

   (a) same side communication (0-1, 1-0, 2-3, 3-2),
   (b) straight communication (0–2, 2–0, 1–3, 3–1),
   (c) cross communication (0–3, 1-2, 2-1, 3–0).

2.2.2. Output Simulation Scenario to Simulate the NiP Architecture Using HXN as IoC. The following assumptions are considered while simulating the NiP architecture using HXN as IoC.

(1) No packet can survive for more than 6 clock cycles where each clock cycle equals 1 for loop.

(2) During same side communication (0-1, 1-0, 2-3, 3-2) priority is given to the chaining link, after that straight path or exchange path accordingly.

(3) Once a packet arrives at SE "2" or "3" it cannot back track. This assumption is made to reduce the network latency. So only "1" disjoint path is made available in form of "3" or "2" accordingly.

(4) The communication in program takes place accordingly as:

   (a) same side communication (0-1, 1-0, 2-3, 3-2),
   (b) straight communication (0–2, 2–0, 1–3, 3–1),
   (c) cross communication (0–3, 1-2, 2-1, 3–0).

2.2.3. Algorithm for Generating Test Cases to Be Used for NiP with PNN and HXN. The run time Complexity of Algorithm 1: test cases is $O(n)^4$.

Input: n, stores Number of parallel processing NoC.
Source, a NoC type array, stores the Source NoC Numbers a part of NoC structure.
Destination, a NoC type array, stores the Destination NoC Numbers a part of NoC structure.
Payload, a part of NoC structure that holds the data generated as Source NoC.
Output: Payload.

(1) Initialize global variable
Set graph[6, 6] = [(0, 1, 1, 1, 0, 0), (1, 0, 1, 1, 0, 0), (1, 1, 0, 1, 1, 1), (1, 1, 1, 0, 1, 1), (0, 0, 1, 1, 0, 1), (0, 0, 1, 1, 1, 0)]
busy[6] = [0, 0 · · · 0]
success[4] = [0, 0, 0, 0]
(2) Form a class packet
Attributes:
PUBLIC header[2]: INTEGER
PUBLIC flit[4]: INTEGERs:
METHOD: PUBLIC packet()
BEGIN
flit[0] = 0
flit[1] = 0
flit[2] = 0
flit[3] = 0
End METHOD
End class packet
packet p[4]
(3) Open the file "pairs.txt" with handle file1
Open the file "pairs1.txt" with handle file2
print(ar)
FOR pairs = 3 to pairs = 4
WHILE (EOF of file1|file2 is not reached)
IF (pairs = 3)
getline from file1 = ar
ELSE
getline from file2 = ar
[call puts with ar]puts(ar)
FOR i = 0 to 2
Input source and destination of NoC packet
[initialize s and d] Set = ar[i ∗ 4] − 48 and d=ar[i ∗ 4 + 2] − 48
(4) IF s <= 2 THEN
[Initialize header]header[0] = s−1
ELSE
header[0] = s
IF d <= 2 THEN
[Initialize header] header[1] = d−1
ELSE
[Initialize header 1] header[1] = d
End FOR
(5) Output header for all packets
Initialize flit and busy FOR all i = 0 to 3
p[i].flit = p[i].header[0] and busy[i].header[i] = 1
(6) FOR k = 0 to [value of pairs]
IF flit = header[] and flit != 99 and flit != 100 THEN
flit = 99
(7) IF graph[flit][header] = = 1 and flit != 99 and flit != 100
THEN
IF busy = = 1 THEN
IF busy[2] = = 0 THEN
Assign flit[0] = 0
Assign busy[2] = 1
Assign flit = 2[0]
(8) IF busy[3] = 0
Initialize flit[0] = 0
Assign busy[3] = 1
Assign flit[0] = 3

Algorithm 5: Continued.

```
(9)  ELSE DO
       Assign flit[0] = 0
       Assign flit[0]t = header[1]
       Assign flit[0] = 1
(10) ELSE IF flit[0] != 9 and flit[0] != 100 THEN
       Initialize flag = 0
        Loop FOR x = 0 to 5
          Set flag = 0
        Loop FOR x = 0 to 5
          IF flit(x) = 1 and busy(x) = 0
             Assign flit[0] = 0
             Assign flit[0] = x
              Assign flit[0] = 1
              Assign flag = 1
(11) IF flag = 0
             Assign flit[0] =100
(12) Print flit[k] FOR k = 0 to [value of pairs]
(13) Assign busy[p[k].flit[3]] = 0
     Assign flit[3] = flit[2]
     Assign busy[p[k].flit[3]] = 1
     Assign flit[2] = flit[1]]
     Assign busy[p[k].flit[3]] = 1
     Assign flit[1] = flit[0]
     Assign busy[p[k].flit[3]] = 1
(14) Loop e = 1 to pairs
       If flit(3) = 99
        increment success[pairs − 1]
(15) Loop z = 1 to z = pairs
     Assign success[z] = success[z]/z
      Print [success[z] and efficiency]efficiency = (success[z]/total[z])∗100

The run time Complexity of the Algorithm: NoC_WS_REGULAR_HXN is O(n)^2
```

Algorithm 5: NoC_WS_REGULAR_HXN.

Table 11: The following source destination pair of NoC is used to show 100% efficiency of the NiP (based on PNN, used as IoC). All source destination pair delivered the information to the respective NoC. The following data does not reflect the parallel communication among the NoCs using packet and wormhole switching algorithms.

Pair 1		PNN-PS status	PNN-WS status
Source NoC	Destination NoC		
0	1	Reached successfully	Reached successfully
0	2	Reached successfully	Reached successfully
0	3	Reached successfully	Reached Successfully
1	0	Reached successfully	Reached successfully
1	2	Reached successfully	Reached successfully
1	3	Reached successfully	Reached successfully
2	0	Reached successfully	Reached successfully
2	1	Reached successfully	Reached successfully
2	3	Reached successfully	Reached successfully
3	0	Reached successfully	Reached successfully
3	1	Reached successfully	Reached successfully
3	2	Reached successfully	Reached successfully

Here PS represents packet switching and WS represents wormhole switching.

Table 12: The following source destination pairs of NoCs are used to show 100% efficiency of the NiP (based on PNN, used as IoC). All source destination pairs delivered the payload information to the respective NoC. The following data does reflect the parallel communication among the NoCs using packet and wormhole switching algorithms.

Pair 1		Pair 2		PNN-PS status	PNN-WS status
Source NoC	Destination NoC	Source NoC	Destination NoC		
0	2	1	3	All reached successfully	All reached successfully
0	1	1	2	All reached successfully	All reached successfully
0	1	2	3	All reached successfully	All reached successfully
0	2	2	3	All reached successfully	All reached successfully
0	2	2	1	All reached successfully	All reached successfully
0	1	3	2	All reached successfully	All reached successfully
0	1	3	0	All reached successfully	All reached successfully
0	2	3	0	All reached successfully	All reached successfully
1	2	0	3	All reached successfully	All reached successfully
1	3	2	0	All reached successfully	All reached successfully
1	2	3	0	All reached successfully	All reached successfully
2	1	0	3	All reached successfully	All reached successfully
2	3	1	0	All reached successfully	All reached successfully
2	0	3	1	All reached successfully	All reached successfully
3	2	0	1	All reached successfully	All reached successfully
3	0	1	2	All reached successfully	All reached successfully
3	1	2	0	All reached successfully	All reached successfully

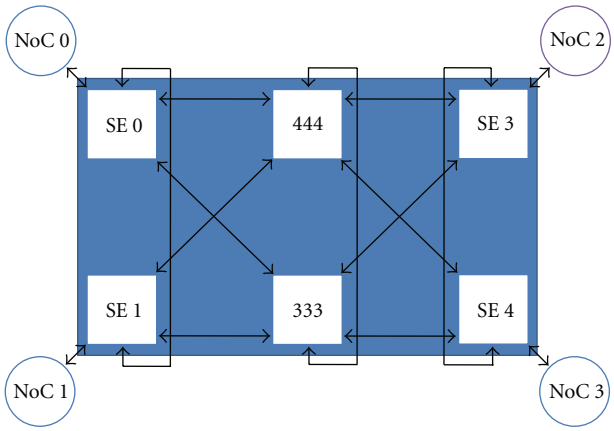


Figure 21: The simulation state of NiP-HXN model at second step of worst case.

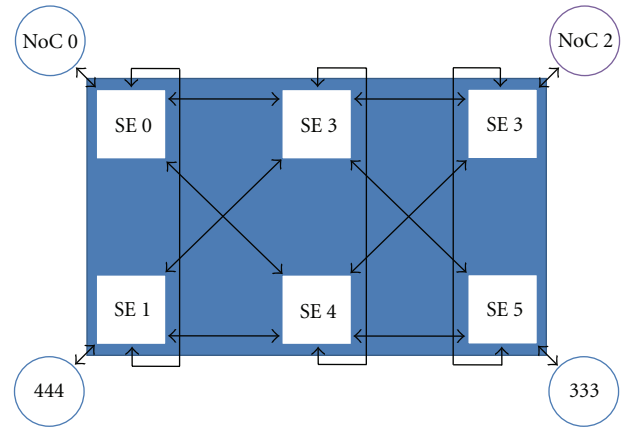


Figure 22: The simulation state of NiP-HXN model at third step of worst case.

Proof of Correctness. (1) For the FOR loops starting on lines #6 to #12, Time $T_1 = (3 \times 3)t_1$ (here t_1 is constant).

(2) For the FOR loops starting on line #13 to #31, Time $T_2 = (12 \times 12 \times 12 \times 12)t_2$ (here t_2 is constant).

Total time:

$$T = T_1 + T_2 = (3^2)t_1 + (12^4)t_2. \quad (1)$$

If we talk about n terms,

$$T = (n^2)t_1 + (n^4)t_2 = n^2 + n^4 = n^4 = O(n)^4. \quad (2)$$

Therefore, Complexity in Big (O) notation is $O(n)^2$. □

2.3. Fault-Tolerant Dynamic Switching Algorithms

2.3.1. Fault-Tolerant Packet Switched Algorithm for Dynamic Communication among NoCs Using Irregular PNN. The run time Complexity of Algorithm 2: NoC_PS_IRREGULAR_PNN is $O(n)^2$.

Proof of Correctness. (1) For the FOR loops starting on lines #2 and #8, Time $T_1 = 4t_1 + 3t_2$ (here t_1 and t_2 are constant).

(2) For the FOR loop starting on line #13, Time $T_2 = n \times t_3$ (here t_3 is constant).

(3) For the FOR loop starting on line #17, Time $T_3 = (n+1) \times n \times t_4$ (here t_4 is constant).

Table 13: The following source destination pairs of NoCs are used to show 62.5% and 72.5% efficiency of the NiP (based on PNN, used as IoC). All source destination pairs delivered the information to the respective NoC. The following data does reflect the parallel communication among the NoCs using packet and wormhole switching algorithms. Some of the payload information are dropped in the network due to nonavailability of paths.

Pair 1		Pair 2		Pair 3		Pair 4		HXN-PS status	HXN-WS status
Source NoC	Destination NoC	Source NoC	Destination NoC	Source NoC	Destination NoC	Source NoC	Destination NoC		
0	3	1	3	2	3	3	1	Pair 2 dropped others reached successfully	Pair 2 dropped others reached successfully
1	2	2	1	3	1	0	1	Pair 2 dropped others reached successfully	Pair 2 dropped others reached successfully
2	1	3	2	1	0	0	1	All reached successfully	All reached successfully
1	3	3	1	2	1	1	2	Pair 1 and pair 4 dropped others reached successfully	Pair 1 dropped others reached successfully
0	2	2	3	1	3	3	1	Pair 3 dropped others reached successfully	Pair 3 dropped others reached successfully
1	3	3	1	0	2	2	0	Pair 1, Pair 3, and pair 4 dropped other reached successfully	Pair 1 and Pair 4 dropped other reached successfully
0	1	1	0	2	3	3	2	All reached successfully	All reached successfully
3	0	3	1	3	2	1	3	Pair 1 and pair 2 dropped other reached successfully	Pair 1 and Pair 2 dropped others reached successfully
0	2	1	3	2	1	3	0	Pair 1, Pair 2, and Pair 3 dropped other reached successfully	Pair 1, pair 2 dropped other reached successfully
0	2	1	3	1	2	0	1	Pair 1 and Pair 2 dropped other reached successfully	Pair 1 dropped other reached successfully

Table 14: Initial source and destination information used to set up communication between NoCs.

Source	Destination	Payload
0	1	111
1	0	222
2	3	333
3	2	444

Table 15: Intermediate SE information while data moves from source NoC to destination NoC at second step of best case.

Source NoC	Destination NoC	Switching element at second step
0	1	0
1	0	1
2	3	4
3	2	5

(4) For the FOR loop starting on line #24, Time $T_4 = 2t_5$ (here t_5 is constant).

$$\begin{aligned}\text{Total time} &= T_1 + T_2 + T_3 + T_4\\ &= 4t_1 + 3t_2 + nt_3 + (n^2 + n)t_4 + 2t_5\\ &= 4t_1 + 3t_2 + nt_3 + n^2t_4 + nt_4 + 2t_5\\ &= 4t_1 + 3t_2 + n(t_3 + t_4) + n^2t_4 + 2t_5\\ &= n^2 = O(n)^2.\end{aligned} \tag{3}$$

Therefore, Complexity in Big (O) notation is $O(n)^2$. □

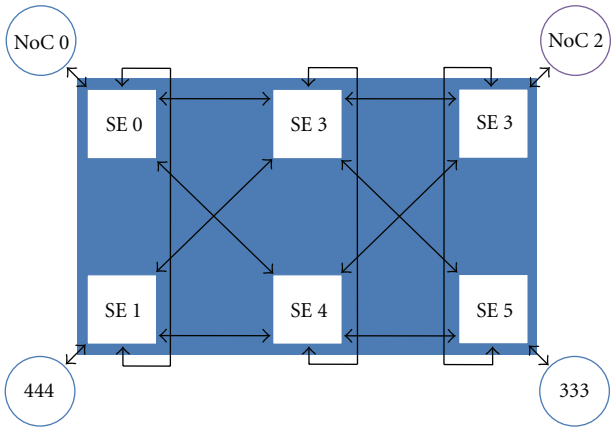


Figure 23: The simulation state of NiP-HXN model at fourth step of worst case.

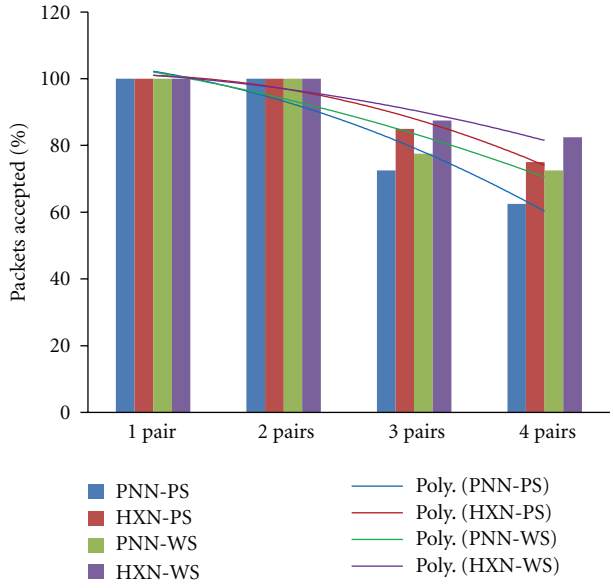


Figure 24: The graph shows the comparison of the number of packets reached to the destination NoC (using the PNN and HXN as IoC) to the number of NoC pairs, amongst which communication is setup.

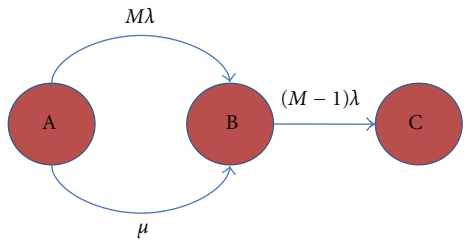


Figure 25: A Markov reliability model for a MIN with repair.

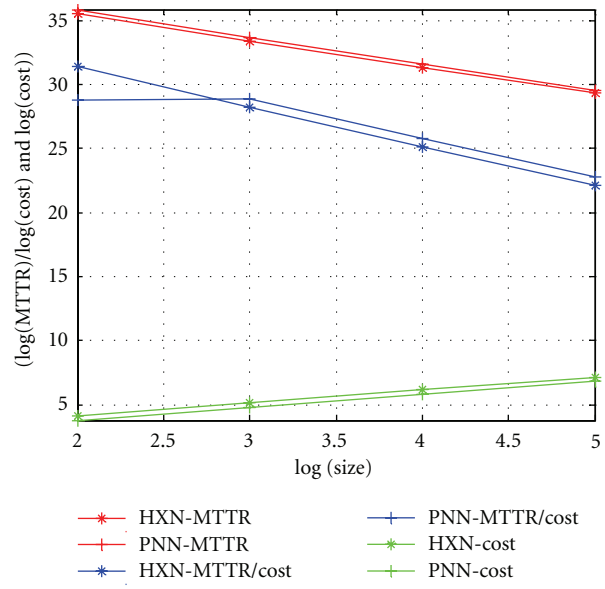


Figure 26: MTTF under repair of PNN and HXN along with the MTTF lower bounds.

Table 16: Intermediate SE information while data moves from source NoC to destination NoC at third step of best case.

Source NoC	Destination NoC	Switching element at third step
0	1	2
1	0	0
2	3	3
3	2	4

Table 17: Intermediate SE information while data is moves from source NoC to destination NoC (in between some information has been reached to the desired destination successfully) at fourth step of best case.

Source NoC	Destination NoC	Status/switching element at fourth step
0	1	1
1	0	Reached successfully at NoC 0
2	3	5
3	2	Reached successfully at NoC 2

Table 18: The payload successfully reached to its desired destination NoC.

Source NoC	Destination NoC	Status
0	1	Reached successfully at NoC 1
2	3	Reached successfully at NoC 3

2.3.2. Fault-Tolerant Wormhole Switched Algorithm for Dynamic Communication among NoCs Using Irregular PNN. The run time Complexity of Algorithm 3: NoC_WS_IRREGULAR_PNN is $O(n)^2$.

Table 19: Initial data used to set up communication between NoCs.

Source	Destination	Payload
0	1	111
0	2	222
0	3	333
3	1	444

Table 20: Intermediate SE information while data is moves from source NoC to destination NoC (in between some information destroyed before reaching to the destination successfully) at second step of worst case.

Source NoC	Destination NoC	Status/switching element at second step
0	1	Destroyed
0	2	Destroyed
0	3	0
3	1	5

2.3.3. Output Simulations of NiP Using PNN as IoC. To show the simulations, two kinds of simulating scenarios are considered here for intra-NoC communication:

(1) best case,

(2) worst case.

Figure 6 shows the architecture of the NiP model in the form of blocks. The same is used for showing the simulation behavior of the intra-NoC communication in NiP for the best and the worst case (Note: Figure 6 to Figure 23 have been designed using the legends declared in Figure 5.).

Best Case. For this communication scenario, the following source NoC and destination NoC have chosen as an example, shown in Table 1.

First Step. Refer to Figure 8 and Table 2.

(1) When communication is set up between NoC 0 and NoC 3 the payload to be sent is 111 therefore the payload first moves to SE 0.

(2) For communication between NoC 1 and NoC 3, the payload to be sent is 222 therefore the payload moves to SE 1.

(3) For communication between NoC 2 and NoC 3, the payload to be sent is 333 therefore the payload moves to SE 3.

(4) For communication between NoC 3 and NoC 1, the payload to be sent is 444 therefore the payload moves to SE 4.

Second Step. Refer to Figure 9 and Table 3.

(1) For communication between NoC 0 and NoC 3, the payload 111 moves to SE 2.

Table 21: Intermediate SE information while data moves from source NoC to destination NoC at third step of worst case.

Source NoC	Destination NoC	Switching element at third step
0	3	3
3	1	2

Table 22: Intermediate SE information while data moves from source NoC to destination NoC at fourth step of worst case.

Source NoC	Destination NoC	Switching element at fourth step
0	3	5
3	1	1

Table 23: The payload successfully reached to its desired destination NoC.

Source NoC	Destination NoC	Status
0	3	Reached successfully at NoC 3
3	1	Reached successfully at NoC 1

(2) For communication between NoC 1 and NoC 3, the payload 222 moves to SE 0.

(3) For communication between NoC 2 and NoC 3 the payload 333 should move to either the SEs 2, 0, 4 (already carrying payloads) otherwise the packet gets destroyed and hence the communication link does not proceed further.

(4) For communication between NoC 3 and NoC 1, the payload 444 moves to SE 1.

Third Step. Refer to Figure 10 and Table 4.

(1) For communication between NoC 0 and NoC 3, the payload 111 moves to SE 4.

(2) For communication between NoC 1 and NoC 3, the payload 222 moves to SE 2.

(3) For communication between NoC 3 and NoC 1, the payload 444 has reached its destination NOC 1 and transferred.

Fourth Step. Refer to Figure 11 and Table 5.

(1) For communication between NoC 0 and NoC 3, the payload 111 has reached its destination NoC 3 and transferred.

(2) For communication between NoC 1 and NoC 3, the payload 222 moves to SE 4.

Fifth Step. Refer to Figure 12 and Table 6.

(1) For communication between NoC 1 and NoC 3, the payload 222 has reached its destination NoC 3 and transferred.

Table 24: The following source destination pair is used to show 100% efficiency of the NiP (based on HXN, used as IoC). All source destination pair delivered the information to the respective NoC. The following data does not reflect the parallel communication among the NoCs using packet and wormhole switching algorithms.

Pair 1		HXN-PS status	HXN-WS status
Source NoC	Destination NoC		
0	1	Reached successfully	Reached successfully
0	2	Reached successfully	Reached successfully
0	3	Reached successfully	Reached successfully
1	0	Reached successfully	Reached successfully
1	2	Reached successfully	Reached successfully
1	3	Reached successfully	Reached successfully
2	0	Reached successfully	Reached successfully
2	1	Reached successfully	Reached successfully
2	3	Reached successfully	Reached successfully
3	0	Reached successfully	Reached successfully
3	1	Reached successfully	Reached successfully
3	2	Reached successfully	Reached successfully

Table 25: The following source destination pairs of NoCs are used to show 100% efficiency of the NiP (based on HXN, used as IoC). All source destination pairs delivered the payload information to the respective NoC. The following data does reflect the parallel communication among the NoCs using packet and wormhole switching algorithms.

Pair 1		Pair 2		HXN-PS status	HXN-WS status
Source NoC	Destination NoC	Source NoC	Destination NoC		
0	2	1	3	All reached successfully	All reached successfully
0	1	1	2	All reached successfully	All reached successfully
0	1	2	3	All reached successfully	All reached successfully
0	2	2	3	All reached successfully	All reached successfully
0	2	2	1	All reached successfully	All reached successfully
0	1	3	2	All reached successfully	All reached successfully
0	1	3	0	All reached successfully	All reached successfully
0	2	3	0	All reached successfully	All reached successfully
1	2	0	3	All reached successfully	All reached successfully
1	3	2	0	All reached successfully	All reached successfully
1	2	3	0	All reached successfully	All reached successfully
2	1	0	3	All reached successfully	All reached successfully
2	3	1	0	All reached successfully	All reached successfully
2	0	3	1	All reached successfully	All reached successfully
3	2	0	1	All reached successfully	All reached successfully

Worst Cases. For this communication scenario, the following source NoC and destination NoC have been chosen as an example, shown in Table 7. This case is only 50% efficient.

First Step. Refer to Figure 13 and Table 8.

(1) When communication is set up between NoC 0 and NoC 2, the payload to be sent is 111 therefore the payload first moves to the SE 0.

(2) For communication between NoC 1 and NoC 3, the payload to be sent is 222 therefore the payload moves to the SE 1.

(3) For communication between NoC 1 and NoC 2, the payload to be sent is 333 therefore the payload moves to the SE 1 and overwrites the residing packet 222. Hence, payload 333 is at SE 1.

(4) For communication between NoC 1 and NoC 0, the payload to be sent is 444 therefore the payload moves to SE 1 and overwrites the residing packet 333. Hence, payload 444 is at SE 1.

Second Step. Refer to Figure 14 and Table 9.

(1) For communication between NoC 0 and NoC 2, the payload 111 moves to SE 3.

(2) For communication between NoC 1 and NoC 3, the payload 222 moves to SE 4.

Table 26: The following source destination pairs of NoCs are used to show 75% and 82.5% efficiency of the NiP (based on HXN, used as IoC). All source destination pairs delivered the information to the respective NoCs. The following data does reflect the parallel communication among the NoC using packet and wormhole switching algorithms. Some of the payload information are dropped in the network due to non-availability of paths.

Pair 1		Pair 2		Pair 3		Pair 4		PNN-PS Status	PNN-WS Status
Source NoC	Destination NoC	Source NoC	Destination NoC	Source NoC	Destination NoC	Source NoC	Destination NoC		
0	3	1	3	2	3	3	1	Pair 3 dropped others reached successfully	All reached successfully
1	2	2	1	3	1	0	1	Reached successfully	Reached successfully
2	1	3	2	1	0	0	1	Pair 4 dropped others reached successfully	Pair 4 dropped others reached successfully
1	3	3	1	2	1	1	2	Pair 1 and pair 2 dropped others reached successfully	Pair 1 and Pair 2 Dropped Others Reached Successfully
0	2	2	3	1	3	3	1	Pair 2 dropped others reached successfully	All reached successfully
1	3	3	1	0	2	2	0	Reached successfully	Reached successfully
0	1	1	0	2	3	3	2	Reached successfully	Reached successfully
3	0	3	1	3	2	1	3	Pair 1 and pair 2 dropped others reached successfully	Pair 2 dropped others reached successfully
0	2	1	3	2	1	3	0	Pair 1 dropped others reached successfully	Pair 1 dropped others reached successfully
0	2	1	3	1	2	0	1	Pair 1 and Pair 2 dropped others reached successfully	Pair 1 and Pair 2 Dropped Others Reached Successfully

Table 27: Values of MTTR of comparative networks.

MIN attributes	MTTR/cost versus size			
	$N = 4$	$N = 8$	$N = 16$	$N = 32$
HXN-MTTR	50004500000	11365795450	2718451087	665418883
PNN-MTTR	62505000000	13891250000	3290625000	801850961.6
HXN-MTTR/cost	2778027778	315716540.3	37756265.1	4620964.465
PNN-MTTR/cost	4464642857	496116071.4	58761160.71	7159383.586
HXN-cost	18	36	72	144
PNN-cost	14	28	56	112

Third Step. Refer to Figure 15 and Table 10.

(1) For communication between NoC 0 and NoC 2, the payload 111 has reached its destination NoC 2 and transferred.

(2) For communication between NoC 1 and NoC 0, the payload 444 has reached its destination NoC 0 and transferred.

2.3.4. Test Cases and Their Statuses for PNN-Packet Switching and PNN-Wormhole Switching. (See Tables 11, 12 and 13).

2.3.5. Fault-Tolerant Packed Switched Algorithm for Dynamic Communication among NoCs Using Regular HXN. The run time Complexity of Algorithm 4: NoC_PS_REGULAR_HXN is $O(n)^2$.

Proof of Correctness. (1) For the FOR loops starting on lines #2 and #8, Time $T_1 = 6t_1 + 4t_2$ (here t_1 and t_2 are constant).

(2) For the FOR loops starting on line #13, Time $T_2 = n \times t_3$ (here t_3 is constant).

(3) For the FOR loops starting on line #17 to #93, Time $T_3 = (n-1) \times n \times t_4 + 3 \times n \times t_5$ (here t_1, t_2 and t_5 are constant).

$$\begin{aligned} \text{Total time} &= T_1 + T_2 + T_3 \\ &= 6t_1 + 4t_2 + nt_3 + (n^2 - n)t_4 + 3nt_5 \\ &= 6t_1 + 4t_2 + n(t_3 - t_4 + 3t_5) + n^2 t_4. \end{aligned} \quad (4)$$

Therefore, Complexity in Big (O) notation is $O(n)^2$. □

Table 28: The comparison of the existing and proposed parallel communication model.

Properties	Existing systems	Proposed model with packet switching	Proposed model with wormhole switching
(1) Latency	High	Low	Very low
(2) Scalability	Low	Can be extended with wide variety but upon certain limits	Can be extended with wide variety but upon certain limits
(3) Performance	Poor	High	Very high
(4) Fault tolerance	Not fault tolerant	The system is single fault tolerant	The system is single fault tolerant
(5) Reliability	As the system is not reliable therefore, the online repairability is not possible	As the system is fault tolerant therefore, it is reliable and hence, online repairability is possible.	As the system is fault tolerant therefore, it is reliable and hence, online repairability is possible
(6) Contention/Sharing	High	Low	Very low
(7) Probability of data acceptance	Very low	The system (the NiP, using HXN and PNN as IoC) has shown 100% efficiency for the single and double pair of communication among the NoC. For quad pair of communication, the system has 75% efficiency for HXN, whereas the same system has 62.5% efficiency for PNN	The system (the NiP, using HXN and PNN as IoC) has shown 100% efficiency for the single and double pair of communication among the NoC. For quad pair of communication, the system has 82.5% efficiency for HXN, whereas the same system has 72.5% efficiency for PNN
(8) Livelock	Yes	No	No
(9) Deadlock	Yes	Not upon certain limits	Very low

2.3.6. Fault-Tolerant Wormhole Switched Algorithm for Dynamic Communication among NoCs Using Regular HXN. The run time Complexity of Algorithm 5: NoC_WS_ REGULAR_HXN is $O(n)^2$.

2.3.7. Output Simulations Based on Test Cases. To show the simulations, two kinds of simulating scenarios are considered here for inter NoC Communication:

(1) best case,

(2) worst case.

Here Figure 7 shows the architecture of the NiP model in the form of blocks. The same is used for showing the simulation behavior of the inter-NoC communication in NiP for the best and the worst case.

Best Cases. For this communication scenario, the following source NoC and destination NoC have chosen as an example, shown in Table 14.

First Step. Refer to Figure 16 and Table 14.

(1) When communication is set up between NoC 0 and NoC 1, the payload to be sent is 111 therefore the payload first goes to the SE 0.

(2) For communication between NoC 1 and NoC 0, the payload to be sent is 222 therefore the payload goes to the SE 1.

(3) For communication between NoC 2 and NoC 3, the payload to be sent is 333 therefore the payload goes to the SE 4.

(4) For communication between NoC 3 and NoC 2, the payload to be sent is 444 therefore the payload goes to the SE 5.

Second Step. Refer to Figure 17 and Table 15.

(1) When communication is set up between NoC 0 and NoC 1, the payload 111 moves to the SE 2.

(2) For communication between NoC 1 and NoC 0, the payload 222 moves to the SE 0.

(3) For communication between NoC 2 and NoC 3, the payload 333 moves to the SE 3.

(4) For communication between NoC 3 and NoC 2, the payload 444 moves to the SE 4.

Third Step. Refer to Figure 18 and Table 16.

(1) When communication is set up between NoC 0 and NoC 1, the payload moves to the SE 1.

(2) For communication between NoC 1 and NoC 0, the payload 222 has its destination reached NoC 0 and transferred.

(3) For communication between NoC 2 and NoC 3, the payload 333 moves to the SE 5.

(4) For communication between NoC 3 and NoC 2, the payload 444 has reached its destination NoC 2 and transferred.

Fourth Step. Refer to Figure 19 and Tables 17 and 18.

(1) For communication between NoC 0 and NoC 1, the payload 111 has reached its destination NoC 1.

(2) For communication between NoC 2 and NoC 3, the payload 333 moves to the SE 3.

Worst Cases. For this communication scenario, the following source NoC and destination NoC have been chosen as an example, shown in Table 19. This case is only 50% efficient.

First Step. Refer to Figure 20 and Table 19.

(1) For communication between NoC 0 and NoC 1, the payload to be sent is 111 therefore; the payload first moves to the SE 0.

(2) For communication between NoC 0 and NoC 2, the payload to be sent is 222 therefore, the payload moves to the SE 0 and it overwrites the residing packet 111. Hence, payload 222 is at SE 0.

(3) For communication between NoC 0 and NoC 3, the payload to be sent is 333 therefore, the payload moves to the SE 0 and it overwrites the residing packet 222. Hence, payload 333 is at SE 0.

(4) For communication between NoC 3 and NoC 1, the payload to be sent is 444 therefore; the payload moves to SE 5.

Second Step. Refer to Figure 21 and Table 20.

(1) For communication between NoC 0 and NoC 3, the payload 333 moves to the SE 3.

(2) For communication between NoC 3 and NoC 1, the payload 444 moves to SE 2.

Third Step. Refer to Figure 22 and Table 21.

(1) For communication between NoC 0 and NoC 3, the payload 333 moves to SE 5.

(2) For communication between NoC 3 and NoC 1, the payload 444 moves to SE 1.

Fourth Step. Refer to Figure 23 and Tables 22 and 23.

(1) For communication between NoC 0 and NoC 3, the payload 333, residing at SE 5, will get transferred to NoC 3.

(2) For communication between NoC 3 and NoC 1, the payload 444, residing at SE 1, will get transferred to NoC 1.

2.3.8. Test Cases and Their Statuses for HXN-Packet Switching and HXN-Wormhole Switching. (See Tables 25, 26, 27 and Figure 24).

3. Results and Discussions

From Figure 24 in case of Packet Switching NiP, using HXN and PNN as IoC has shown 100% efficiency for the single and double pair of communication among the NoCs. However, as the number of pairs increases, that is, for quad pair of communication, the system using PNN as IoC shows 62.5% of efficiency whereas, the same system using HXN as IoC shows 75% efficiency only. In case of wormhole switching NiP, using HXN and PNN as IoC has shown 100% efficiency for the single and double pair of communication among the NoC. However, as the number of pairs increases, that is, for quad pair of communication, the system using PNN as IoC shows 72.5% of efficiency, whereas the same system using HXN as IoC shows 82.5% efficiency only.

3.1. Mean Time to Repair of IoC. In fault-tolerant MIN, it is always expected that the detection of a fault in SE initiates the repair of the fault, to protect the SE from the occurrence of a second, extremely harmful, fault. Only the conservative approximation of the Mean Time to Failure (MTTF) of single fault-tolerant MIN assumes the repair of the faults. Let the constant failure rate of individual switches be λ and the constant repair rate be μ. Now consider a MIN with M switches and N as network size. For a single fault-tolerant network, the Markov chain model is shown in Figure 25. A Markov chain describes at successive times the states of a system. At these times, the system may have changed from the state it was in the moment before to another or stayed in the same state. The changes of state are called transitions. The Markov property means that the system is memory less, that is, it does not "remember" the states it was in before, just "knows" its present state and hence bases its "decision" to which future state it will transit purely on the present, not considering the past. Here the Markov chain model is represented with three conservative states: state A represents the no-fault state; state B represents the single-fault state, while state C is the two-fault state. The IoC network can tolerate more than one fault. Now it is assumed that if the MIN reaches State C, it has failed. Since the schemes presented here can tolerate more than one faulty switch in many cases, this model should give a lower bound for the MTTF of the system [9]:

$$^{\Phi}\text{MTTF with repair} = \frac{1}{(M-1)\lambda} + \frac{(M-1)\lambda + \mu}{(M-1)N\lambda^2}. \quad (5)$$

Let $\lambda = 10^{-6}$/hour and $\mu = 10^{-6}$/hour (which are typical values used for the calculation). In Figure 26 the MTTR for HXN and PNN is examined for sizes from 4×4 to 32×32 and values are tabulated in Table 27. From graph, it seems that with increase in the size of the MIN, the MTTR improvement factor is actually decreasing. This is due to the conservative assumption that exactly one fault is successfully tolerated. In reality, with increasing size of the MIN the average number of faults tolerated increases. Moreover, it is depicted that the cost of the HXN is higher in comparison to the PNN. As the size goes more higher, that is, the order of the 512×512 or even more the comparison between the cost increases more. Nevertheless, the number of faults tolerated by the HXN is higher in comparison to the number of faults tolerated by the PNN.

4. Conclusion

This paper presents a new method that allows global communication between NoCs in NiP. For this, four $O(n)^2$ fault-tolerant parallel algorithms have been proposed. It allows different NoCs to communicate in parallel using either fault-tolerant irregular PNN or fault-tolerant regular HXN. These two are acting as an IoC in NiP and can tolerate faults also. The two NiP architectures have been automated and the simulation results are provided in the Tables 11–13 and 24–26.

In case of packet switching the NiP, using HXN and PNN as IoC has shown 100% efficiency for the single and double pair of communication among the NoC. However, as the number of pairs increases, that is, for quad pair of communication, the system using PNN as IoC has shown an efficiency of 62.5%, whereas the same system while using HXN as IoC has shown an efficiency of 75% only.

In case of wormhole switching the NiP, using HXN and PNN as IoC has shown 100% efficiency for the single and double pair of communication among the NoC. However, as the number of pairs increases, that is, for quad pair of communication, the system using PNN as IoC has shown an efficiency of 72.5%, whereas the same system while using HXN as IoC has shown an efficiency of 82.5% only.

The comparison of IoC on cost and MTTR concluded that the HXN has the higher cost than the PNN, but the MTTR values of the HXN are low in comparison to the PNN. This signifies that the ability to tolerate faults and online repairing of the HXN is high and faster than the PNN. The various features of the old system and the current proposed system are tabulated in Table 28. From the provided data, one can easily compare the systems and their efficiency. The proposed system comes out to be superior in every aspect.

References

[1] T. Y. Feng, "A survey of interconnection networks," *IEEE Computer*, vol. 14, no. 12, pp. 12–27, 1981.

[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, NY, USA, 1984.

[3] G. B. Adams, D. P. Agrawal, and H. J. Siegel, "A survey and comparison of fault-tolerant multi-stage interconnection networks," *IEEE Computer*, vol. 20, no. 6, pp. 14–27, 1987.

[4] L. N. Bhuyan, "Special issue on interconnection networks for parallel and distributed computing," *IEEE Computer Magazine*, vol. 20, no. 6, 1987.

[5] H. J. Siegel, *Interconnection Network for Large Scale Parallel Processing: Theory and Case Studies*, McGraw Hill, New York, NY, USA, 1990.

[6] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, Tata McGraw-Hill, New Delhi , India, 2000.

[7] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach*, Morgan Kaufmann, San Francisco, Calif, USA, 2003.

[8] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, San Francisco, Calif, USA, 2004.

[9] Nitin, "Component Level reliability analysis of fault-tolerant hybrid MINs," *WSEAS Transactions on Computers*, vol. 5, no. 9, pp. 1851–1859, 2006.

[10] Nitin and A. Subramanian, "Efficient algorithms and methods to solve dynamic MINs stability problem using stable matching with complete ties," *Journal of Discrete Algorithms*, vol. 6, no. 3, pp. 353–380, 2008.

[11] Nitin, S. Garhwal, and N. Srivastava, "Designing a fault-tolerant fully-chained combining switches multi-stage interconnection network with disjoint paths," *Journal of Supercomputing*, vol. 55, no. 3, pp. 400–431, 2011.

[12] Nitin, R. Vaish, and U. Shrivastava, "On a deadlock and performance analysis of ALBR and DAR algorithm on X-Torus topology by optimal utilization of cross links and minimal lookups," *Journal of Supercomputing*, vol. 59, no. 3, pp. 1252–1288, 2012.

[13] Nitin and D. S. Chauhan, "Stochastic communication for application-specific networks-on-chip," *Journal of Supercomputing*, vol. 59, no. 2, pp. 779–810, 2012.

[14] Nitin and D. S. Chauhan, "Comparative analysis of traffic patterns on k-ary n-tree using adaptive algorithms based on Burton Normal form," *Journal of Supercomputing*, vol. 59, no. 2, pp. 569–588, 2012.

[15] M. Sgroi, M. Sheets, A. Mihal et al., "Addressing the system-on-a-chip interconnect woes through communication-based design," in *Proceedings of the 38th ACM IEEE Design Automation Conference*, pp. 667–672, June 2001.

[16] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[17] K. Lee, S. J. Lee, D. Kim et al., "Networks-on-chip and networks-in-package for high-performance SoC platforms," in *Proceedings of the 1st IEEE Asian Solid-State Circuits Conference (ASSCC'05)*, pp. 485–488, Hsinchu City, Taiwan, November 2005.

[18] L. Benini and G. De Micheli, *Networks on Chips: Technology and Tools*, Morgan Kaufmann, San Francisco, Calif, USA, 2006.

[19] S. Murali and G. De Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs," in *Proceedings of the 41st Design Automation Conference*, pp. 914–919, June 2004.

[20] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 71–121, 2006.

[21] T. A. Dumitras, *On-chip stochastic communication [M.S. thesis]*, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pa, USA, 2003.

[22] D. N. Jayasimha, B. Zafar, and Y. Hoskote, *On-Chip Interconnection Networks: Why They Are Different and How to Compare Them*, Intel Corporation, 2006.