Hindawi Publishing Corporation
VLSI Design
Volume 2009, Article ID 941701, 15 pages
doi:10.1155/2009/941701

Research Article

Networks-On-Chip Based on Dynamic Wormhole Packet Identity Mapping Management

Faizal A. Samman,[1, 2] Thomas Hollstein,[1] and Manfred Glesner[1]

[1] *Institute of Microelectronic Systems, Darmstadt University of Technology, Karlstr 15, 64283 Darmstadt, Germany*
[2] *Department of Electrical Engineering, Hasanuddin University, Jl. Perintis Kemerdekaan km.10, Makassar 90245, Indonesia*

Correspondence should be addressed to Faizal A. Samman, faizal.samman@mes.tu-darmstadt.de

Received 7 August 2008; Revised 1 December 2008; Accepted 7 January 2009

Recommended by Rached Tourki

This paper presents a network-on-chip (NoC) with flexible infrastructure based on dynamic wormhole packet identity management. The NoCs are developed based on a VHDL approach and support the design flexibility. The on-chip router uses a wormhole packet switching method with a synchronous parallel pipeline technique. Routing algorithms and dynamic wormhole local packet identity (ID-tag) mapping management are proposed to support a wire sharing methodology and an ID slot division multiplexing technique. At each communication link, flits belonging to the same message have the same local ID-tag, and the ID-tag is updated before the packet enters the next communication link by using an ID-tag mapping management unit. Therefore, flits from different messages can be interleaved, identified, and routed according to their allocated ID slots. Our NoC guarantees in order and lossless message delivery.

1. Introduction

According to the International Technology Roadmap for Semiconductors (ITRS) [1], by the end of this decade, the feature size of a transistor will be 45-nm and it operates below one volt. SoCs will grow to 4-billion transistors running at 10 GHz. The major challenge for SoC designers would be to provide reliable operation of the interacting components. A limiting factor for the performance and, possibly, energy consumption will be presented by on-chip physical interconnections [2].

Most SoC design approaches use some available (intellectual property IP) components to implement an integrated circuit for a certain system application (IP-based design and reuse). A sophisticated communication structure is needed for inter-IP data exchange. Rather than using single wires for single communication between two IPs, or point-to-point communication, a concept of shared segmented communication infrastructures is proposed. Networks-on-chip provide advanced intellectual properties (IPs) communication concepts for systems-on-chip (SoC). The NoC concept has a potential to provide sustainable platforms and a new paradigm in SoC architecture and multiprocessor systems [3].

Figure 1 shows an example of an NoC platform with a 4×4 mesh topology. There are four main components, that is, mesh routers, network interfaces, resources (R), and communication links. Each mesh router is connected via local port with one resource through a network interface. The other ports (East, North, West, and South) are connected with adjacent mesh routers through communication links. Resources can be an IP core or an embedded bus-based platform with one or more processing elements. The resource can also be a memory element with a direct memory access controller. In the context of reconfigurable computing, the NoC platform provides scalable bandwidth with flexible resource-to-resource communication configuration.

Several NoCs that have been developed are NOSTRUM [4], SoCBUS [5], RAW [6], Hermes [7], HiNoC [8], OCTAGON [9], GEXSpidergon [10], SPIN [11], DSPIN [12], NoC by Bartic et al. [13], PNoC [14], Xpipes [15], and ASNoC [16]. All the NoCs have different characteristics and services provided to transfer messages leading to different strategies to design their router microarchitecture.

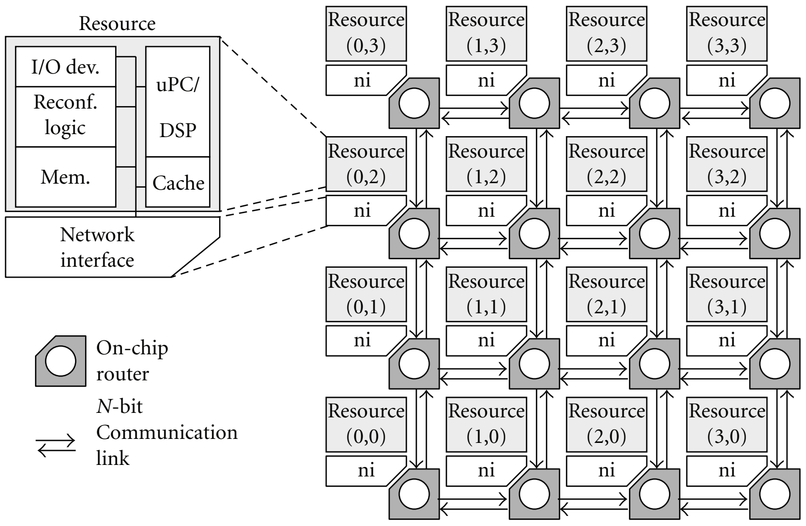


Figure 1: A 2-dimension mesh 4×4 NoC topology.

Asynchronous NoCs are introduced in CHAIN [17], ASPIDA [18], PROTEO [19], and ANoC [20], while in MANGO [21], an asynchronous clock-less NoC is proposed. Asynchronous communication design is a promising concept, but lacks of industrial standard support, especially with respect to testability issues. In synchronous designs, global clock-trees are distributed, which lead to electromagnetic interference effects and clock power consumption.

In this paper, a reconfigurable NoC with a synchronous parallel pipeline router architecture called *XHiNoC* is proposed. XHiNoC stands for eXtendable Hierarchical NoC, and is an extended version of HiNoC [8], which is based on a flexible, extendable design environment. The XHiNoC is developed based on synthesizable modular VHDL objects. Some flexible object modules can be selected and combined with base modules to obtain a specific mesh router prototypes in accordance with a desired specification such as NoC with multicast service [22]. This paper will only focus on an XHiNoC prototype with connectionless unicast service. Some mesh routers have been prototyped by using a static XY routing algorithm and adaptive routing algorithms in a regular 2D mesh topology.

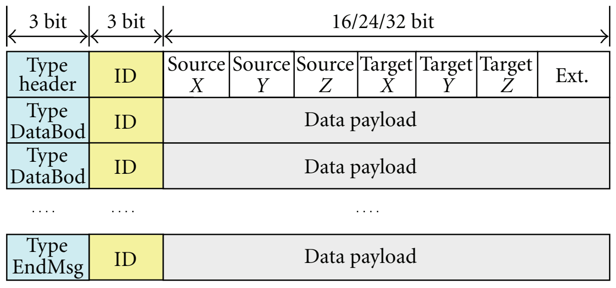
The rest of this paper is organized as follows. Section 2 describes the main contribution of XHiNoC multiplexing technique based on the dynamic local ID-tag mapping management and its related works with time-divison multiplexing methodology, such as Æthereal [23]. Section 3 presents interconnection configuration used by XHiNoC and the comparison with other NoCs in FPGA platforms. In this section, the XHiNoC communication setup is described in detail. In Section 4, selected routing algorithms and link multiplexing technique are introduced. Section 5 describes the microarchitecture of the XHiNoC. Performance evaluation of the XHiNOC under four selected traffic scenarios is exhibited in Section 6. Logic synthesis results using an FPGA device and a 130-nm CMOS standard-cell technology and direct comparison of the performance-area tradeoff between XHiNoC and Æthereal are presented in Section 7. Finally, Section 8 concludes the paper.

2. Related Works and Contributions

Data communication between IP components in the NoC-based systems can be made using circuit-switching, packet (store-and-forward) switching, wormhole, or virtual cut-through switching methods. The circuit-switching that is based on time-division multiplexing (TDM) method has been used by some NoC proposals such in NOSTRUM [4], SoCBUS [5], DSPIN [12], PNoC [14], and Æthereal [23]. Æthereal NoC uses a *slot table* to avoid contention on a link, divide up bandwidth per link between connections, and switch data to the correct output. Every slot table T has time slots S and router outputs N. There is a logical notion of synchronicity, where all routers in the network are in the same fixed-duration slot. In a slot s at most one block of data can be read/written per input/output port. In the next slot, $(s + 1)\% S$, the read blocks are written to their appropriate output ports. Thus, the blocks propagate in a store-and-forward fashion. However, this methodology has disadvantage as explained in the following paragraph according to Figure 3(a).

In Figure 3(a), three packets (A, B, and C) are attempt to set up connections. Four snapshots of the network at successive times are presented. The setup packet A enters node $(2, 2)$ from North (N) input port, and setup packet B enters node $(2, 1)$ from West (W) input port as shown in Figure 3(a)(I). The script (**A:1**) means that packet A will be programmed in time-slot 1 in the next router, and (**B:2**) has the same meaning as well.

As shown in Figure 3(a)(II), packet A has been forwarded to South output port of node $(2, 2)$, and the time-slot 1 is allocated for packet A coming from N. While packet B has also been in South output port of node $(2, 1)$, and the time-slot 2 is allocated for packet B coming from W input port. A bold line shows the progress of the connection setup over



(a)

Type of flit	bin/dec code
No flit	000 (0)
Packet header	001 (1)
Message/data body	010 (2)
End of message	011 (3)

(b)

Figure 2: Packet addressing format for XHiNoC.

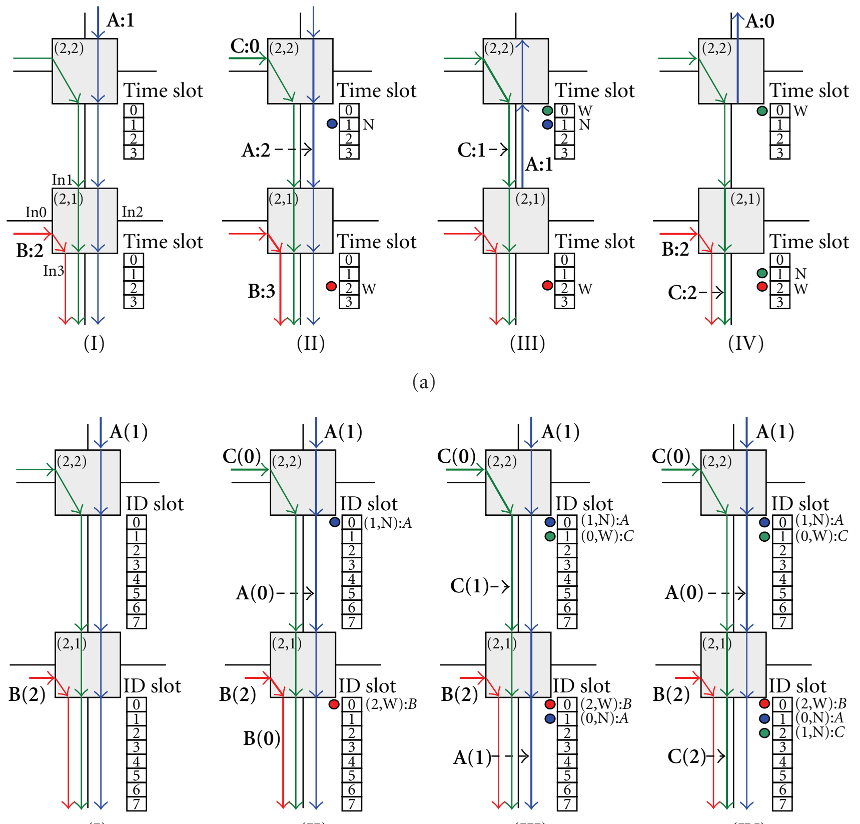


Figure 3: Communication links setup method by Æthereal (Redesign from [23]) and XHiNoC.

time. In every snapshot, the Setup packets are routed to their next link, and the slot table is incremented by one. Thus, in the next router, packets A and B will be programmed in time-slots 2 (**A:2**) and 3 (**B:3**), respectively.

In Figure 3(a)(III), packet A cannot reserve slot 2 for South output port (S) of node $(2, 1)$ because it has been reserved for the connection of packet C, thus the connection setup of packet A fails. Therefore, packet A is routed back along its path to remove the reservations made so far (see Figure 3(a)(III)). In Figure 3(a)(IV), packet A has removed the reservation of slot 1 that it made in Figure 3(a)(II).

The previous explanation has presented the disadvantage of using *time-division multiplexing* (TDM) switching method by Æthereal NoC. In Figure 3(a)(III), there are still three free time-slots in South output port of node $(2, 1)$, that is, time-slot 0, 1, and 3. However, Packet A cannot use that free time-slots, because packet A has been programmed to reserve time-slot 2, that has been reserved by packet B from West (W) input port. Therefore, we propose a more optimistic approach by introducing *ID-tag mapping management* (*IDM*) unit to optimize dynamically the link bandwidth utilization. The *IDM* also consists of *ID-slot Table* and has the same functionality as *slot table* used by Æthereal.

The communication link setup by XHiNoC is presented in Figure 3(b). Once again, four snapshots of the network at successive times are depicted in the figure. For the sake of simplicity, only the ID slot table of the *IDM* in South output port is presented for both mesh nodes. In Figure 3(b)(I), a packet header A with ID-tag 1 ($\mathbf{A(1)}$, numerical value in the bracket represents ID-tag) enters node $(2,2)$ from North (N) input port, while a packet header B with ID-tag 2 ($\mathbf{B(2)}$) enters node $(2,1)$ from West (W) input port.

In Figure 3(b)(II), the packet header A is routed to South (S) output port and is allocated to ID slot 0. Therefore, all payload flits having ID-tag 1 from N input port in node $(2,2)$ will get new ID-tag 0. In Node $(2,1)$, packet header B is routed to S output port and is allocated by *IDM* to ID slot 0. Thus, the South *IDM* unit in this node will map all flits having ID-tag 2 from West (W) input port to receive new ID-tag 0. The bullets in the figure indicate that the ID slots are being used (not free).

In Figure 3(b)(III), a packet header C coming from W input port with ID-tag 0 is routed to the S output port. The South *IDM* unit maps packet C into ID slot 1. Hence, each payload flit having ID-tag 0 from W input port in node $(2,2)$ will get new ID-tag 1. While packet A coming from N with ID-tag 0 is allocated by the South *IDM* unit in node $(2,1)$ to ID slot 1. Hence, it receives new ID-tag 1 before being forwarded from node $(2,1)$ into the next node. Figure 3(b)(IV) describes also the same mechanism, where packet C in node $(2,1)$, which is coming from N input port with ID-tag 1, will be mapped to receive the new ID-tag 2.

If a header flit requests a new ID slot allocation to reserve link bandwidth, then the *IDM* unit in each output port will search for a free ID-tag. After finding a new free ID-tag, the *IDM* unit will identify and record the current ID of the header and from which port it comes. Therefore, each payload flit belonging to the same message (because of having the same ID-tag) will be mapped by the *IDM* unit to receive the new ID-tag by using ID-based look-up table mechanism. Therefore, the XHiNoC approach is more optimistic than Æthereal in terms of optimal link bandwidth allocations.

3. Interconnection and Links Setup

A message in XHiNoC is associated with a single packet. For a unicast message, the packet will have only one header flit, even if the size of the message is very large. The packet format is presented in Figure 2. The 38-bit packet consists of a header flit followed by payload flits. Two additional 3-bit heads are type and (Identity ID) bits. The flit type can be as header (1), message body (2), and the end of message body or the last flit of the message (3). Only one header flit is asserted for one message (as one packet), even if the size of the message is extremely large.

The 3D source and target address of the packet are asserted in the header flit. The source and target Z are dedicated for addressing the resource tiles of subnetworks (e.g., in tree-based topologies) when our NoC will be extended to be a hierarchical NoC. The subnetworks will be connected to a local port of each mesh node. Hence, each resource tile located in the subnetwork will have (X,Y,Z) address, where the (X,Y) denotes the address of the mesh network, and the (Z) represents the address of the tile in the subnetwork. Each flit belonging to the same message has the same local identity number (ID-tag) to differentiate it from other flits of different messages when it passes through a communication link.

The local ID-tag of each data flit of a packet will vary over different communication links in order to provide a scalable concept. Dynamic variations of the message ID-tag will be controlled and mapped by *IDM* units allocated at each output port. This ID-tag will vary over the network links to support a wire sharing methodology, which is described in Section 4.2.

3.1. Interconnect Reconfiguration. Mapping an application onto an NoC-based multiprocessor system-on-chip (MPSoC) can be divided into pre- and postmanufacture mappings. In the premanufacture mapping, interconnection between processing elements is successively configured after obtaining optimized configuration data from an optimized mapping problem. The data represent an interconnect configuration map that is used to implement the required wire interconnection.

FLUX interconnnect [24], for instances, uses this approach where on demand interconnections are established before or during program excecution. The proposed reconfigurable interconnections look more like point-to-point connections. Therefore, the required number of links increases exponentially related to the number of processing elements. Certainly, the FLUX network cannot support the postmanufacture implementation.

In the postmanufacture approach, link interconnects have been wired in advance. Therefore, one possible way to reconfigure the wire interconnects is by implementing slot tables (programmable registers) over the on-chip router which can be programmed by users to configure the required interconnections between the processing elements.

An FPGA-based NoC by Bartic et al. [13] uses look-up tables that have an entry for each IP in the NoC, and their content can be dynamically changed by an OS at runtime. PNoC [14] on FPGA fabric supports also a dynamic nodes removal and insertion at runtime in the NoC system via routing table updates. Certainly, the drawback of both is that the required memory space for the routing tables grows proportionally with the number of IPs.

Another approach is by introducing a parallel programming model in software level to reconfigure the interconnections. In the parallel programming model, node addresses of all processing elements or IPs are introduced. The users will program the NoC by using a data parallel, multithread, distributed shared-memory, or message passsing programming model. The compilation result of the parallel program will result in executable object files that will be written in the instruction memories of each selected processing element (PE) that will run the parallel program.

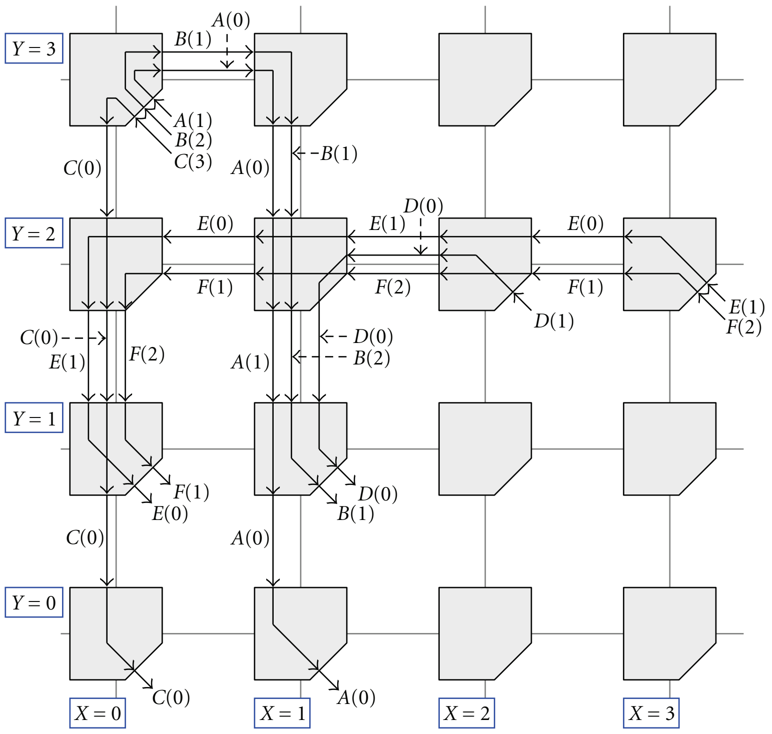


Figure 4: Communication links setup based on dynamic wormhole packet identity.

Selecting the optimal object file code allocation for the selected PE is preceeded by an optimal mapping problem with constraints, for example, bandwidth requirements to transfer data between two PEs or IPs. Heuristically, various routing functions can be involved in this optimal problem to find out whether the constraint can be satisfied or not. The interconnections between processsing elements are then set up locally at runtime by on-chip routers using static or adaptive routing algorithms based on source-target address information present in the header of packets being sent.

The postwired application mapping is not only suitable for ASIC implementations, but also can be used for NoC-based MPSoC applications running on FPGA platforms. The interconnect reconfiguration in the XHiNoC is made available by using runtime programmable ID-slots in a routing table at each incoming port and in an ID-tag management unit at each outgoing port. The working organization between the routing table and the ID-tag management units has enabled a flexible data multiplexing over the on-chip network.

Initially, the routing tables and ID-slots of the ID-management units are empty, which means that an interconnect from source to target node has not been configured. The links interconnection can be configured by sending a header flit containing source and target addresses with a certain local ID-tag. When a header flit is detected at the output of an FIFO buffer, then a routing logic unit will compute a routing direction. The routing direction is then copied into an ID-slot of the routing table in accordance with the local ID-tag number presents in the header. Hence, payload flits having the same ID-tag with the header injected from the source node will be routed to the similar path (see Figure 4). In contrast, the links interconnection can be closed by sending a flit with *EndMsg* type (see again Figure 2) to remove the assignment and routing data from the routing tables and from the ID-tag management units. Before entering the next downstream communication channel, the local ID-tag of the header and payload flits must be updated to support flexible data multiplexing. Section 3.2 will present how to set up and schedule the communication interconnects at runtime.

3.2. Communication Links Setup. An example of a communication link setup of a wormhole message packet based on the dynamic local ID-tag mapping management in the XHiNoC interconnection network is shown in Figure 4. The (X, Y) address of each mesh node is denoted in the left and bottom sides of the figure. We assume that all messages are routed using a static XY routing algorithm. But certainly, the communication link setup process is valid for static and adaptive routing methods.

Messages A, B, and C with ID-tag 1, 2, and 3, respectively, are injected from a processor in node $(0, 3)$. Messages E and F with ID-tag 1 and 2, respectively, are injected from a processor in node $(3, 2)$, while message D with ID-tag 1 is injected from node $(2, 1)$. Each flit belonging to the same message from the same processor will be injected with the same ID-tag.

The ID-tag of each flit belonging to the same message will vary over the communication links and can be controlled and mapped by the *IDM* unit at each output port of the mesh router node. For instance, the flits of message A have ID-tag 0 after passing East (E) output port of mesh node $(0, 3)$ and

IDM unit East output port node (0,3)

ID state	ID	(Old ID, from)	Msg.
●	0	(1, L)	A
●	1	(2, L)	B
	2		
	3		
	4		
	5		
	6		
	7		

● ID tag is in use state.

(a)

IDM unit South output port node (0,2)

ID state	ID	(Old ID, from)	Msg.
●	0	(0, N)	C
●	1	(0, E)	E
●	2	(1, E)	F
	3		
	4		
	5		
	6		
	7		

● ID tag is in use state.

(b)

IDM unit South output port node (1,2)

ID state	ID	(Old ID, from)	Msg.
●	0	(0, E)	D
●	1	(0, N)	A
●	2	(1, N)	B
	3		
	4		
	5		
	6		
	7		

● ID tag is in use state.

(c)

IDM unit West output port node (2,2)

ID state	ID	(Old ID, from)	Msg.
●	0	(1, L)	D
●	1	(0, E)	E
●	2	(1, E)	F
	3		
	4		
	5		
	6		
	7		

● ID tag is in use state.

(d)

LUT local input port node (0,3)

ID	Dir.	Msg.
0	E	A
1	E	B
2	S	C
3		
4		
5		
6		
7		

(e)

LUT West input port node (1,3)

ID	Dir.	Msg.
0	S	A
1	S	B
2		
3		
4		
5		
6		
7		

(f)

LUT East input port node (1,2)

ID	Dir.	Msg.
0	S	D
1	W	E
2	W	F
3		
4		
5		
6		
7		

(g)

LUT North input port node (0,1)

ID	Dir.	Msg.
0	S	C
1	L	E
2	L	F
3		
4		
5		
6		
7		

(h)

LUT North input port node (1,1)

ID	Dir.	Msg.
0	L	D
1	S	A
2	L	B
3		
4		
5		
6		
7		

(i)

Figure 5: Routing and ID-mapping management tables of the *LUT* and *IDM* units, respectively, according to Figure 4. ID-tag is in use state; E: East; N: North; W: West; S: South; L: Local port.

South (S) output port of node $(1, 3)$ successively, then ID-tag 1 after passing S output port of mesh node $(1, 2)$. Afterwards they have new ID-tag 0 after passing S output port of mesh node $(1, 1)$. Finally, the message flits receive new ID-tag 0 after ejecting from the on-chip network via local (L) output port of mesh node $(1, 0)$.

In our NoC infrastructure, different messages can be injected from a processor in a overlapping-time. As shown in Figure 4, for instance, the processor connected to node $(0, 3)$ injects three different messages, that is, messages A, B, and C. The messages are destinated to target node $(1, 0)$, $(1, 1)$, and $(0, 0)$, respectively. A flit injected with ID-tag 1 (belonging to message A) will always arrive target node $(1, 0)$ and a flit with ID-tag 2 (belonging to message B) will arrive target node $(1, 1)$ as well as flits injected with ID-tag 3 (belonging to message C) will also always arrive target node $(0, 0)$. It means that a small group of flits of a message can be injected in overlap injection time with other groups of the other flit messages.

If the processor injects a last flit indicating the end of packet body (its ID-tag is the same as ID-tag of the packet body and header), then the last flit will close the connection that has been set up in advance by the packet header and get the reserved local IDs free. Afterwards, the free local ID-tag can be then used by other messages injected from that processor.

Figures 5(a), 5(b), 5(c), and 5(d) present the ID-tag mapping tables of the selected *IDM* units in output ports. For instance, *IDM* unit at South output port at node $(0, 2)$ (see Figure 5(b)) has allocated its three new ID slots for three different messages (i.e., messages C, E, and F). This *IDM* unit has still five available ID slots. For example, message F coming from East (E) input port with current ID-tag 1 will be mapped into ID slot 2. It means that each flit with ID-tag 1 coming from E input port (flits belong to message F) will receive new ID-tag 2 before passing South output port of node $(0, 2)$.

Figures 5(e), 5(f), 5(g), 5(h), and 5(i) represent the routing tables of the selected *LUT* units in input ports. For instance, the *LUT* unit at local input port in node $(0, 3)$ has recorded the routing directions of messages A, B, and C. The direction recording is undertaken after the *RHL* has computed the routing direction based on target node address stated in the header flit of each message. The direction is then saved in the routing table of the *LUT*. For example, packet A associated with ID-tag 0 has EAST (E) routing direction. It means that each flit with ID-tag 0 (flits belong to message A) will be routed to EAST direction.

4. Routing Algorithms and Switch Multiplexing

4.1. Exchangeable Routing Engine Modules. The XHiNoC microarchitecture is developed based on a modular approach. Some modules can be exchanged and instantiated to form a new on-chip router prototype with a specific characteristic. The routing engine (RE) is a reconfigurable unit, which consists of combination of *router hardware logic* (*RHL*) and *router look-up table* (*LUT*) units. The modular RHL units enable us to easily design a new on-chip router

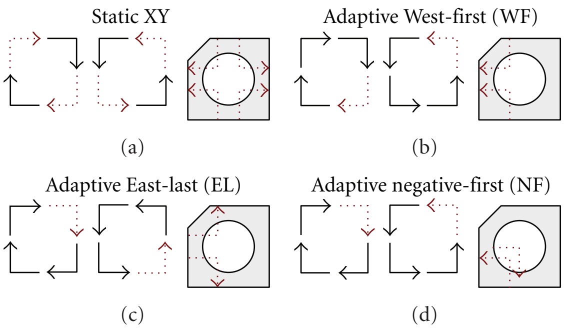


Figure 6: The turn model of the selected routing algorithms.

```
Xoffs=Xdest−Xsource;
Yoffs=Ydest−Ysource;
          ......
          ......
elseif Xoff>0 and Yoffs>0 then
    if NumOfusedID(NORTH)<NumOfusedID(EAST) or
        (NumOfUsedID(NORTH)=NumOfUsedID(EAST)
         and NumOfDataInFIFO(NORTH)<NumOfDatain
         FIFO(EAST)) then
           route=north;
    else route=east; end if;
          ......
          ......
end if;
```

Algorithm 1: Adaptivity for making North or East routing.

prototype with a specific routing function implemented on those units. Five optional routing algorithms proposed in this paper, which are realized in the *RHL* module to evalute the NoC performance, are static XY routing, adaptive *West-first* (WF), *negative-first* (NF) [25, 26], *odd-even (OE) turn* [27], and *East-Last* (EL) routing algorithms.

4.1.1. Static Routing Algorithm. Figure 6(a) shows the turn model of XY static routing. The dotted lines denote the prohibited turns, while the solid lines denote the allowed turns. The diagram in the right side of each figure shows the turn models when they are mapped into a mesh node diagram. The turn models are introduced to avoid cyclic dependency (to avoid deadlock). In mesh router prototype with XY routing algorithm, message packet will always be routed firstly in X (horizontal) direction, and then into Y (vertical) direction. Therefore, turns from North to East, North to West, South to East, and South to West are prohibited as depicted in Figure 6(a).

4.1.2. Adaptive Routing Algorithm. Figures 6(a), 6(c), and 6(d) depict the turn model of the selected adaptive routing algorithms. Some existing minimal adaptive routing algorithms are used, where adaptiveness of the routing selection depends on two signals. When there are two possible directions to route a packet, then the router will consider firstly the number of free ID slot of two possible output links. The packet will be routed to the direction, where more free ID slots are available. This runtime routing adaptivity is also used by AdNoC [28] where routing decisions are made locally depending on available bandwidth in each direction to the neighboring router.

The main difference between AdNoC and XHiNoC is that XHiNoC does not use virtual channels. The use of virtual channels will increase logic gates consumption. Our routing algorithms are also simpler resulting in a routing logic unit with less than 300 logic gates per router in comparison with 2877 logic gates comsumed by a routing logic unit of AdNoC. The XHiNoC routing logic will also consider the number of free registers from the FIFO buffers in the next two adjacent mesh nodes. If the numbers of free ID slots are the same for two possible directions, then the router will select a direction, where more free registers are available in the next output link. If both values are the same, the routing logic will prefer a nonturn direction. Algorithm 1 presents a cutsection of an adaptive routing algorithm to make a NORTH or EAST routing adaptively.

(1) West-First (WF) Routing Algorithm. The turns from North to West and South to West are prohibited in WF routing algorithm. Hence, packets are routed firstly to West when the target nodes are located in North-West or South-West quadrants. Packets can be adaptively routed when destination nodes are located in North-East or South-East quadrants.

(2) East-Last (EL) Routing Algorithm. In EL routing algorithm, the turns from East to North and East to South are prohibited. Hence, packets are routed to East at last when the target nodes are located in North-East or South-East quadrants. Packets can be adaptively routed when destination nodes are located in North-West or South-West quadrants.

(3) Odd-Even (OE) Routing Algorithm. The adaptive routing with odd-even turn model selected in our prototype uses combination of adaptive WF and EL routing algorithms. In the even column, the routers use EL adaptive routing, while in the odd column, the routers uses WF adaptive routing. However, attention must be payed for the routing adaptiveness in an even column. In an odd column, packet cannot be routed to East when target node is located in North-East or South-East quadrant of the next East column (even column), because in the even column, the rule for EL routing is regarded, for example, East to North and East to South turns are prohibited.

(4) Negative-First (NF) Routing Algorithm. In NF routing algorithm, the turns from West to South and South to West are prohibited. Hence, packets are routed firstly to negative direction when the target nodes are located in South-East (Y-direction firstly) or North-West (X-direction firstly) quadrants. Packets can be adaptively routed when

destination nodes are located in North-East or South-West quadrants.

4.2. Dynamic Packet Identity Management. Figure 7 presents how message packets can be mixed or interleaved in the same FIFO buffer by using our proposed local ID-tag mapping management method with wormhole packet switching. The packet interleaving method is proposed for a fair output link access and is very effective to tackle a deadlock problem in a multicast data delivery [22]. For the sake of simplicity, only *LUT* units of *RE* modules at West (*W*) and East (*E*) input ports in node (2, 3) as well as *LUT* unit at North (*N*) input port in node (2, 2) are presented. And only the *IDM* unit at South (*S*) input port in mesh node (2, 3) is shown. Packet *A* with ID-tag 3 enters mesh node (2, 3) from the *E* input port, while packet *B* with ID-tag 2 comes from the *W* input port.

As shown in Figure 7, we assume that the header of each packet has been forwarded, and the *RHL* unit in the *RE* module has computed the routing direction and recorded it into the *LUT* register address based on its ID-tag number. Hence, in mesh node (2, 3), flits coming from *W* input port with ID-tag 2 will always be routed to South output port as well as flits coming from *E* input port with ID-tag 3 will be also routed to South (see East and West *LUT* in mesh node (2, 3)).

The *IDM* unit manages the link bandwidth utilization and will guarantee that different flits of different packets will not have the same ID-tag before entering the next downstream mesh node. Thus, the *LUT* unit in the next input port can route incoming flits with service based on their ID-tags. As shown in Figure 7, the South *IDM* in mesh node (2, 3) maps packet *A* into ID slot 0 and packet *B* into ID slot 1. Hence, packets *A* and *B* will receive new ID-tag 0 and 1 respectively after leaving for mesh node (2, 3). Packets *A* and *B* are interleaved in mesh node (2, 2) and are then routed by the *LUT* unit at North input port based on their ID-tags. Flits with ID-tag 0 (belonging to message packet *A*) will be routed to South direction, while the flits with ID-tag 1 (belonging to message packet *B*) will be routed to East.

5. XHiNoC Microarchitecture

5.1. General Mesh Switch Router Architecture. The XHiNoC mesh router prototypes are developed using modular synthesizable VHDL approach. The modular-based design enables us to exchange some modules with other modules allowing a partial architecture reconfiguration at design time. The architecture is scalable to follow a required specification. For instance, if we need only unicast or both unicast and multicast support [22], then some modules can be easily exchanged and inserted or removed without making a hard or large modification.

Some modules in the XHiNoC are public units, which are commonly used for many required prototypes. And the others are flexible/exchangeable units, which can be replaced by other units to obtain a certain router prototype. The *router hardware logic* (*RHL*), for instance, as shown in Figure 8 is a exchangeable module. A routing algorithm is implemented in this module. The selected routing algorithms that are used in this paper are described in Section 4.1.

One routing engine is allocated at each incoming port allowing up to 5 parallel simultaneous crossbar connections. This approach will increase bandwidth capacity of the router. Indeed, because our router architecture in this paper is made for mesh NoC topology, then the bandwidth capacity of the NoC can be scaled well.

The architecture of the NoC router can be classified into three main groups that is, *logical modules* in each port, a *crossbar switch*, and a centralized LCFS (*link controller and flow supervisor*) module. For the sake of simplicity, only components in the West port are presented in the figure. In each port, there are some components such as an *FIFO buffer*, a *routing eingine*, (*RE*), an *ID-tag mapping manager* (*IDM*) unit, and *link state controller* (*LSC*).

The *RE* module consists of a combination of the *RHL* and *router look-up table* (*LUT*) to support runtime routing adaptivity and in-order delivery guarantee when using a deadlock-free adaptive routing algorithm. The *RHL* computes the required routing direction based on the target address information stated in a header flit and the underlying routing algorithm. The routing direction of the message is then copied into the routing table registers of the *LUT* and is indexed based on its ID-tag. Thus, each flit belonging to the same message packet (because it has the same ID-tag) will be routed to the same direction. By using that combination, an extra reconfiguration unit to program the slot table of the LUT unit is not required. The *RE* is allocated for each input port to improve bandwidth capacity of each router (*routing parallelism*).

Because of using a parallel pipelined wormhole packet switching, the XHiNoC is equipped with network traffic flow control. The use of the network flow control enables variable determination of the number of registers in the FIFO buffer (*FIFO depth*). The LSC is a state machine, which controls the flow of a flit from the output port into the next downstream FIFO buffer. If the next FIFO is full, then the (link state controller LSC) unit will not let the flit enter the next FIFO, until one space in the register of the next FIFO is free. The *central LCFS* will be explained in Section 5.2. The functionality of the *IDM* unit has been presented in Section 2. Sections 3.2 and 4.2 have also explored how flits of different messages can be interleaved in a communication link.

5.2. Link Controller and Flow Supervisor. The LCFS functionalities are to control the links in the crossbar switch and to supervise neighbour congestion states. The structure of the LCFS is depicted in Figure 9. It consists of *direction assignment*, *arbiter*, *winner flit encoder* (*EncWin*), and *multiplexor* (*Mux1*) units. The last tree modules are assigned for each output port to enable parallel outport access which can increase the bandwidth capacity of each router. The LCFS receives routing direction requests from all routing engine (*RE*) units, and a full flag from the network interface and the neighbor nodes.

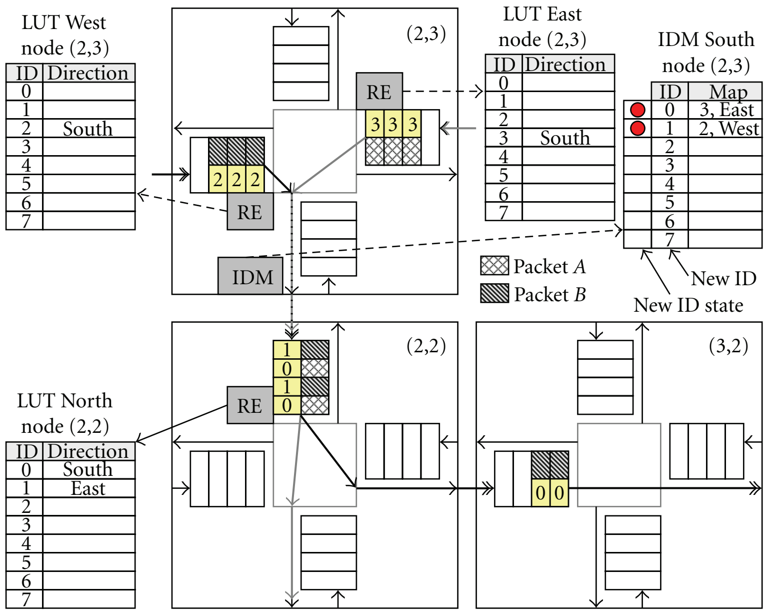


Figure 7: Dynamic wormhole packet identity-tag mapping management controlled by *LUT* and *IDM* modules.

The *direction assignment* unit is used to assign the routing directions from all *RE* units in the input ports, to distribute properly the routing direction request signals (EAST, NORTH, WEST, SOUTH, LOCAL) and to decode the signals from 3-bit into 1-bit signals. These 1-bit signals are then applied to the input ports of an *arbiter* unit. The *arbiter* is in charge of selecting a winner flit of all flits requesting the same output port. By detecting the high-level values of those signals, then the *arbiter* will select a winner which has right to access the output port. This mechanism is realised by applying per flit round robin arbitration to support a fair mechanism to access and to share the output links (see Section 4.2). If the FIFO in the next node is full, then the arbiter will not select a winner to access the requested ports (*link-level flit flow control*).

The *EncWin* encodes the winner signals and passes these signals to the *Mux1* units in all output ports. Signal interconnect from output ports of the *EncWin* units into input ports of the *Mux1* units is aimed to control the flit traffic flows in the *crossbar switch* (*cbswitch*) module. The *Xout* signals from *EncWin* units are connected to the *cbswitch* module to select a flit that will be forwarded to an output port. The *Mux1* unit is in charge of granting the FIFO that holds the winner flit to access the output port, thus the data flit can be released from the FIFO queue.

6. Performance Evaluation

6.1. Selected Traffic Scenarios. The performance of the proposed XHiNoC prototypes with different selected routing algorithms as described in Section 4.1 is evaluated by using four different traffic scenarios as shown in Figure 10. In the matrix-transpose traffic scenarios as depicted in Figures 10(a) and 10(b), 6 different packets are injected into 6 different nodes. While in the complement traffic scenarios as depicted in Figures 10(c) and 10(d), 8 different packets are injected into 8 different nodes. Source and target nodes are identified by alphabets S and T, respectively, followed by numerical alphabet.

Each injected packet consists of 128 flits (1 header flit followed by 127 payload flits). Therefore, in the transpose traffic scenarios, the total number of 768 flits (6 × 128) is injected in the network, while in the complement traffic scenarios, the total number is 1024 flits (8 × 128). Each message is well encoded, so that it can be easily recognised in the networks. Each flit of the messages is then numbered in-order, thus it is easy for us to check packet-loss, packet integrity, and out-of-order delivery problem.

Certainly, the total number of required clock cycles for packet transfer heavily depends on the NoC traffic load. But, the purpose of these traffic scenarios is not only to measure the number of required cycles to transfer the last flit of each message packet under different traffic situations and different routing algorithms, but also to observe the effectiveness and the efficiency of using the local ID-tag mapping management technique to schedule the link at runtime. The measurement starts from injection time of the header flits from source node until ejection time of the last flit of each packet in the target node. The start time of the packet injections is the same at every node, where after some cycles, the congestion flags can probably trace back to the injection nodes, because of the existing link sharing in the network for instance. Afterwards, the injection rates can be automatically controlled by network interfaces with similar mechanism as the flit flow control at link-level.

6.2. Performance Measurement Results. Figure 11 shows the table of total cycle requirements to transfer the last flit of each

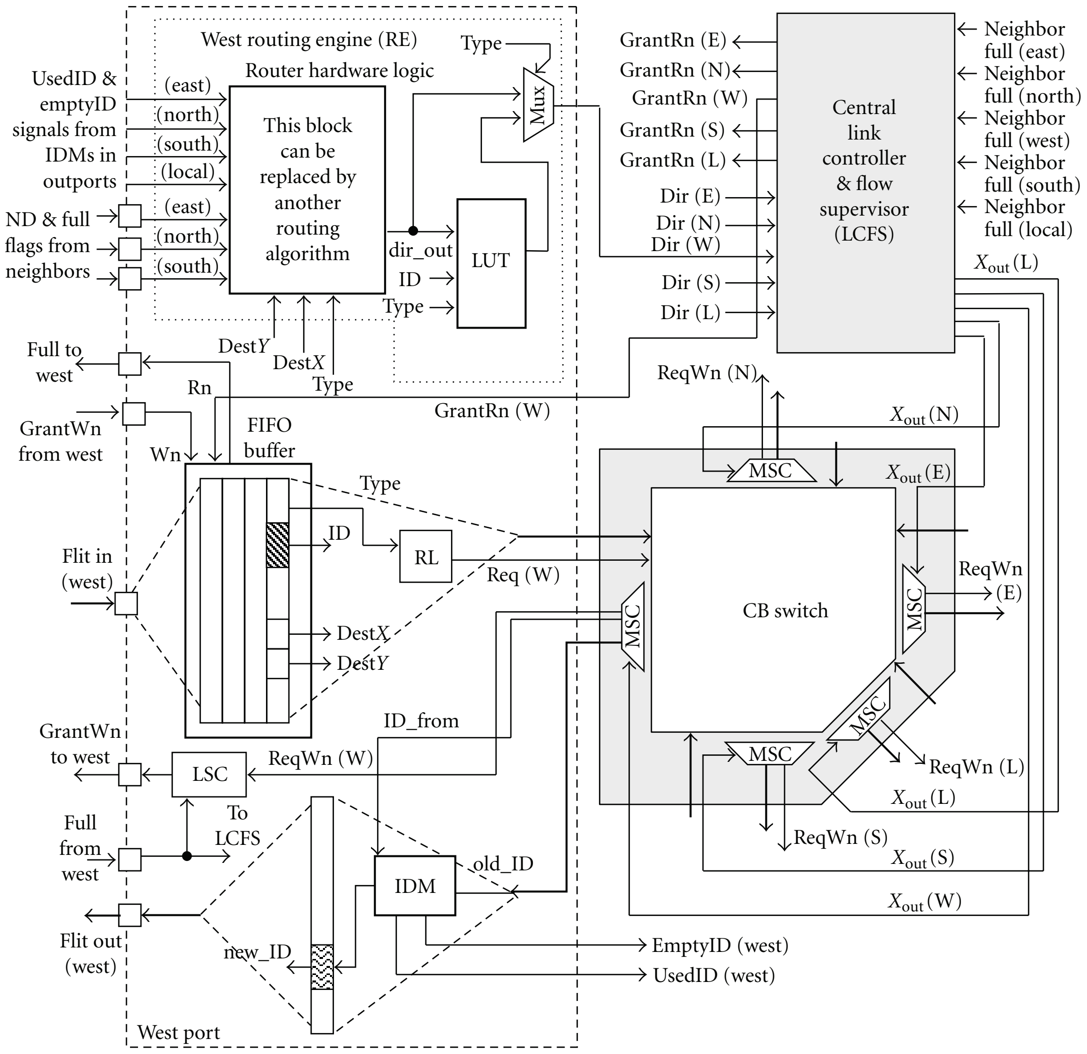


FIGURE 8: Generic architecture of the XHiNoC mesh router.

packet from source nodes to target nodes. Figure 12 presents the average cycle requirements to transfer the last flit of each packet for the five selected routing algorithms. The average number of cycles is calculated by summing the total cycle to transfer the last flit of each packet divided by number of packet. In this case, a number of 6 and 8 packets are injected in the transpose and complement traffic scenarios, respectively.

It looks that, the routing algorithm, which has the best performance, differs from each traffic scenario. In the transpose traffic 1, the WF, OE, and NF adaptive routing functions show the best performance. In the transpose traffic 2, the best performance is shown by the EL and NF adaptive routing functions. In the complement traffic 1, the static XY routing shows the best performance. While in the complement traffic 2, the performance of the static XY, the WF, and EL adaptive routing functions is the best. The performance of each routing algorithm is strongly dependent on the traffic condition.

In general, the RTL simulation experiments have proved the effectiveness and the efficiency of using the local ID-tag mapping management to schedule the messages with wormhole packet switching technique. The methodology has successfully allocated one available ID-slot per communication link for each message at runtime. Therefore, as long as the bandwidth requirement does not exceed the maximum capacity of each communication link (as represented by the available ID-slots), all flits of the messages can be scheduled and transmitted from source to destination nodes.

TABLE 1: Synthesis results of the routers on a Virtex2 FPGA device (flit size: 32 + 6 bits, FIFO buffer depth: 8).

Routing Al.	XY	WF	EL	OE	NF
Number of slices	4884	5009	5016	5017	5114
Max. freq. (MHz)	83.39	91.02	89.43	88.57	100.51

7. Logic Synthesis

7.1. Synthesis on FPGA. The mesh router prototypes have been synthesized on VirtexII-Pro (target device xc2vp30).

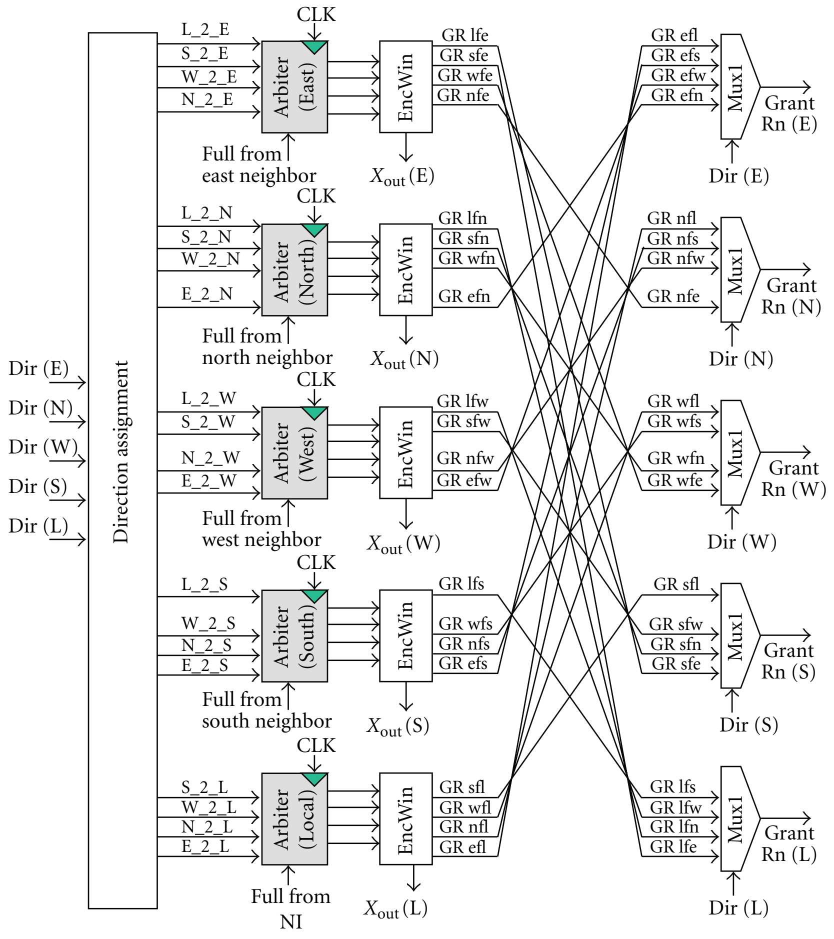


Figure 9: Detail structure of the LCFS.

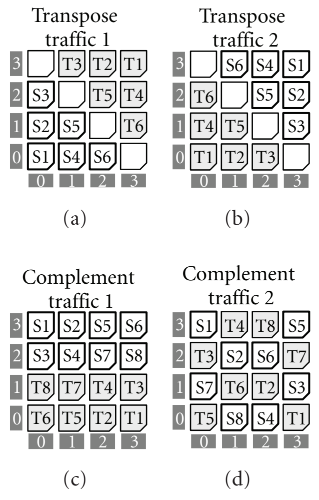


Figure 10: Selected traffic scenarios.

Table 1 shows the number of slices required to synthesis five mesh router prototypes with different routing algorithms as well as maximum working frequency. The table shows also area overheads to implement adaptive routing over static routing algorithm. The implementation of (*negative-first NF*) routing algorithm gives the largest area, for example, 5114 slices and an overhead of 4.71% over static routing.

Table 2: Number of slices of the router with WF routing on a Virtex2 FPGA device.

Flit size	FB depth = 4	FB depth = 8	Overhead
16 + 6 bits	2809	3662	30.37%
24 + 6 bits	3136	4380	39.67%
32 + 6 bits	3468	5009	44.43%

Table 2 presents the comparison number of slices by varying the bit size of the packet flit and the depth of FIFO buffer (the number of registers in the FIFO buffer). For instance, if the bit size of the payload flit (words size) is double increased (from 16 bits to 32 bits), then the number of required slices increases 23.46% when FIFO depth is 4 and increases 36.78% when FIFO depth is 8.

Table 2 shows also the overhead percentage of slices number when the FIFO depth is increased from 4 to 8 registers. For instance, when the flit size is 32 + 6 bits (32-bit words size), then the logic consumption will increase 44.43% if the FIFO depth is increased from 4 to 8 registers.

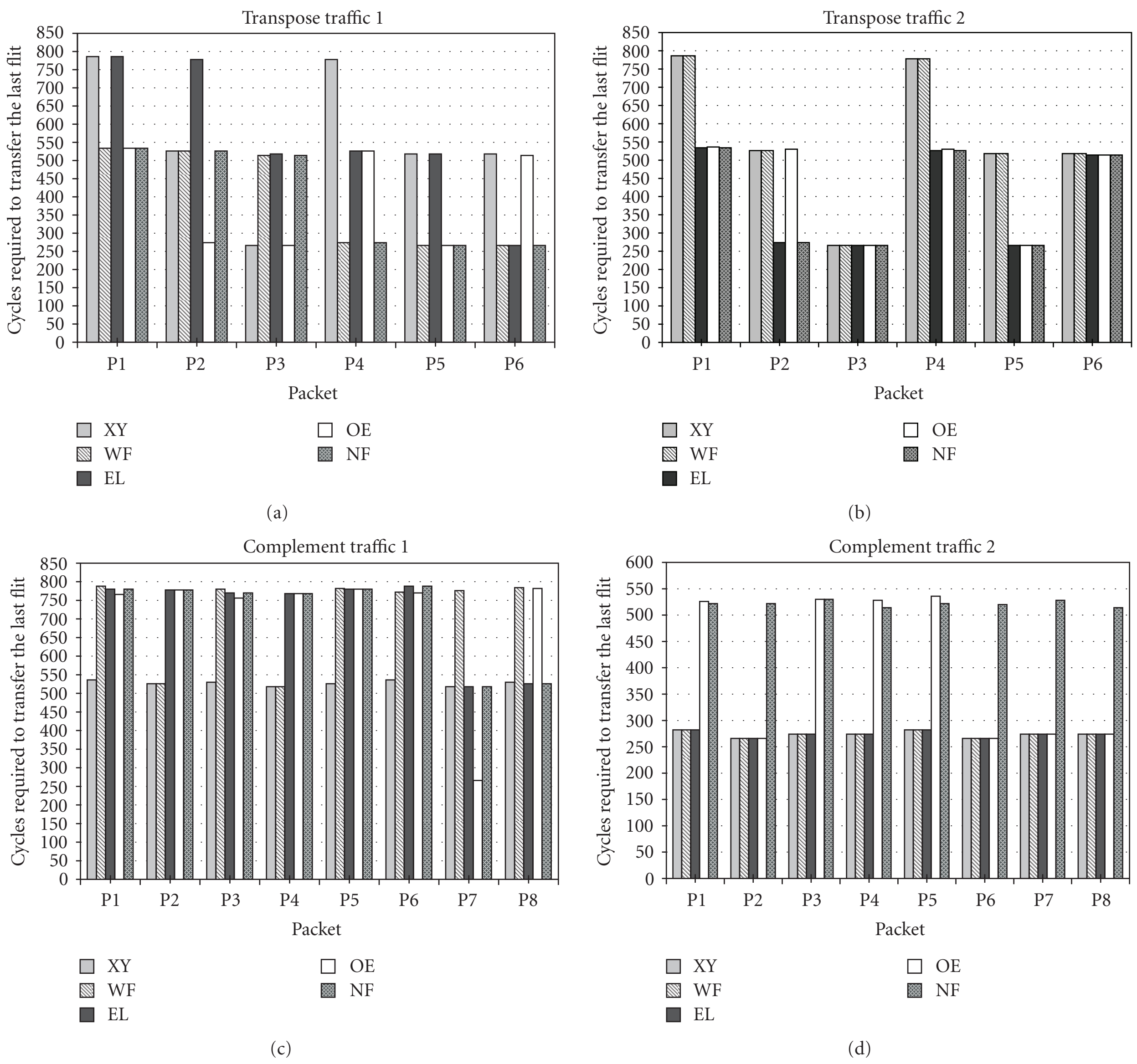


Figure 11: Total cycle requirements to transfer last flits.

Table 3: Logic synthesis of router prototypes with adaptive routing algorithms (flit size: 32 + 6 bits, FIFO depth: 4).

Router's routing Alg.	WF	OE	EL	NF
Num. of logic cells	7149	7132	7119	7206
Total cell area (mm^2)	0.1058	0.1057	0.1054	0.1064

Meanwhile, the logic consumptions increase 30.37% and 39.67%, if the word sizes are 16 bits and 24 bits, respectively. The results show that FIFO buffers dominate logic gates consumption.

The irregular circuit-switched PNoC [14] with 8 IO-ports, 32-bit data width, consumes 1305 slices and can be clocked at 126 MHz. The PNoC consumes less slices than XHiNoC, because XHiNoC uses ID-tag management units and link-level flow control. Furthermore, PNoC uses a dynamic module replacement via routing table updates that is suitable for FPGA implementation, but not for ASIC, except that additional reconfiguration unit to update the contents of the routing tables is provided.

7.2. Synthesis using CMOS Standard-Cell.

7.2.1. Synthesis Data. Table 3 presents the logic area evaluation of mesh router prototypes with adaptive routing algorithms after synthesis using 130-nm CMOS standard-cell technology from *United Microelectronics Corporation (UMC)*. In Table 3, total numbers of logic cells and cell areas of four NoC prototypes with different routing algorithms and the same FIFO buffer depth (4 registers) are presented. It looks that the variation of the logic and area consumptions of the adaptive mesh router prototypes is very small.

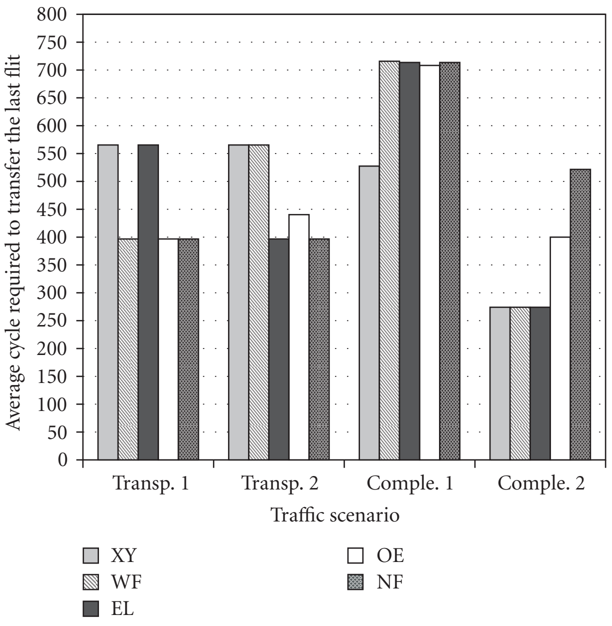


Figure 12: Average last flits transfer latency.

Table 4: Logic synthesis of the router prototype with static XY routing algorithm (flit size: 32 + 6 bits) using UMC 130-nm and 180-nm standard-cell technologies.

130-nm techn.	FIFO depth: 2	FIFO depth: 4
Num. of logic cells	5363	6661
Total cell area (mm^2)	0.0767	0.1018
Max. Freq. (MHz)	472	453
180-nm techn.	FIFO depth: 2	FIFO depth: 4
Num. of logic cells	5033	6572
Total cell area (mm^2)	0.123	0.168
Max. freq. (MHz)	264	247

Table 5: Power estimation at 200 MHz with UMC 180-nm technology (1.8 V). (Flit size: 32 + 6 bits, FIFO depth: 4.)

Routing Alg.:	XY	WF	NF
Net switching power (mW)	7.49	7.94	8.04
Cell internal power (mW)	63.98	60.87	66.43
Cell leakage power (uW)	0.89	0.99	0.84

Table 4 exhibits the logic cell consumptions, total cell areas, and maximum working frequencies for NoC prototype with static XY routing algorithm by varying the FIFO buffer size (2 and 4 registers). Increasing the depth of FIFO will not only increase the logic consumption but also degrade the maximum working frequency. The synthesis results present the synthesis data using 130-nm and 180-nm technologies. The migration from 180-nm to 130-nm technology will increase the maximum working frequency and reduce the estimated logic area. Table 5 presents also the estimation of power dissipation of the static and adaptive routers using 180-nm standard-cell technology.

Figure 13 represents a circuit layout of the mesh router prototype using static XY routing algorithm with 4-depth FIFO buffer. The cell area of the IDM units is highlighted in the circuit layout. The standard-cell place and route are made using *silicon encounter* tool from *Cadence* and 180-nm standard-cell library from *UMC*. In the future, we will layout the overall NoC-based on-chip multiprocessor using the circuit layout prototype. We are now in progress to develop the programming model of the on-chip multiprocessor using our XHINoC interconnect platform.

7.2.2. Direct Comparison with other TDM-based NoCs. By using a 130-nm standard-cell technology, the logic area of Æthereal on-chip router [23] is 0.2600 mm^2 (a queue depth of 8 flits of 3 words of 32-bit). By using the same feature size technology our XHiNoC router with static routing algorithm has total cell area of 0.0767 mm^2 if the FIFO depth is 2, and 0.1018 mm^2 if the FIFO depth is 4. While the XHiNoC routers with adaptive routing and 4-depth FIFO have total cell areas of about 0.106 mm^2. The area of Æthereal router is mainly due to the use of virtual output channels to buffer *best-effort* and *guaranteed-throughput* packets in different FIFO buffers.

The maximum frequency to transfer data in Æthereal router (32-bit word size) is 500 MHz resulting in an aggregate bandwidth of 5×500 MHz $\times$ 32 bits = 80 Gbit/s. While the aggregate bandwidth of the XHiNoC router (static routing, 2-depth FIFO, 32-bit word size) is 5×472 MHz $\times$ 32 bits $\times$ 1/2 = 37.76 Gbit/s. The use of a routing engine with combined router hardware logic and routing look-up table gives contribution to the smaller maximum data frequency compared to the maximum data frequency of the Æthereal. The XHiNoC aggregate bandwidth is divided by two because of the use of two stage cycle pipeline data transmission.

When only a routing table was used to implement the routing engine, there is still a potentiality to increase the maximum data frequency of our NoC. In this case, an additional reconfiguration unit to schedule the link at compile time is needed. Even if the ID-based slot allocation is done at compile time, the optimal ID-based slot allocation does not required a global network view. Computing an optimal time-based slot allocation for all connections at compile time (as used by Æthereal) requires the global network view and may be expensive [23].

NOSTRUM NoC [4] has reported that its router consumes 13896 equivalent NAND gates (independent from the standard-cell technology). Without reporting the logic area, SoCBUS NoC [5] can be clocked at 1.2 GHz in a 180-nm technology process. The DSPIN NoC router [12] with 90-nm technology has gate area of about 0.082 mm^2 after gate-level synthesis (4-depth (guaranteed-service GS) queue, 8-depth (best-effort BE) queue, 34-bit flit size). On a 500 MHz implementation, each GS channel in DSPIN has a bandwidth of 8 Gbit/s (40 Gbit/s for 5 GS channels). The logic area and data frequency of DSPIN compared with our XHiNoC are approximately the same with similar 130-nm technolgy size.

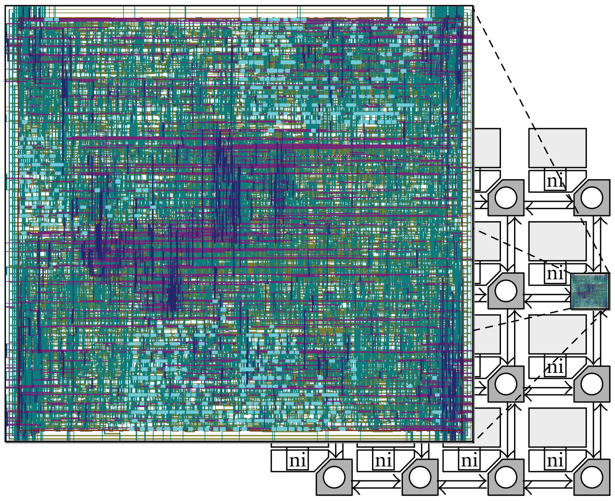


FIGURE 13: The circuit layout of the router prototype using 180-nm UMC technology with static XY routing and 4-depth FIFO buffer.

8. Conclusions

Our XHiNoC prototypes with the local ID-tag mapping management technique have shown a good performance to serve wormhole packets and to schedule link interconnects locally in each router at runtime. As explained in Section 2 before, the dynamic local ID-tag mapping management used by our XHiNoC is more flexible than the TDM-based circuit switching used by Æthereal [23]. The XHiNoC's ID-based scheduling is easier and more effective to reconfigure links interconnection both at runtime and compile time than the TDM-based scheduling used by Æthereal. Unfortunately, the work in [23] did not show an experiment to verify the TDM-based circuit switching by using an example of traffic scenario.

In our experiment, all traffics can be accepted in the target nodes for all traffic scenarios using static and adaptive routing algorithms. There is no flit-loss, and all traffics are accepted in order, because even if adaptive routing is used, only the header flit is routed adaptively to find optimal link. Payload flits will follow the links that have been reserved by the header flits using wormhole switching.

Message delivery communication services can be divided into *connectionless* (*best-effort*) and *connection-oriented* (*guaranteed-bandwidth*) communication. In the connection-oriented communication, a header flit must be injected from a source node firstly to reserve links in the network. After finding connection to its destinated node, a response flit will be sent back to the source node. After the response flits arrive the source node, payload flits start being injected from source node. Hence, this approach is also called the *guaranteed-bandwidth* service, because the packet will not be injected to the network before a guarantee exists, that is, one slot bandwidth of each link connecting source and target nodes has been reserved for the packet.

Our recent XHiNoC implementation uses connectionless communication, where messages are sent like UDP packets in internet world. Therefore, the optimal resource placement in the NoC platform should be undertaken, and the result must guarantee that there will be no communication links, which are consumed exceeding their maximum capacity in a certain period of time. In this case, the maximum capacity is related to available ID slots. Otherwise a message must wait for other messages until one of them has closed the reserved link or release one ID slot to be free.

The optimal problem could be undertaken, because traffics are predictable in the context of SoC application. If the solution of above optimal problem cannot be found, then we must increase the number of available ID slots. In our recent XHiNoC implementation, we use 3 bits for ID slot identification. It means that there are 8 available ID slots. By increasing ID tag bits to 4, 5 or, 6 bits, there will be 16, 32, or 64 ID slots available for link bandwidth consumption, respectively.

The use of IDM units in our proposed ID-tag-based multiplexing technique has given a significant contribution to the logic consumption. The logic cell area of the IDM units (e.g., in router with 4-depth FIFO and static routing) is about 36% of the total logic cell area. Since the critical path of router is found in the FIFO buffer and in the routing engine unit, the IDM unit does not affect the maximum allowed data frequency. However, there is an additional data pipeline at the outgoing port in order to let the IDM unit to update and to map the old and the new IDs of each flit. Hence, flits flowing through the network router will experience additional latency of one cycle period. Because of the additional data pipeline, the latency will increase proportionally to the number of hops.

The current XHiNoC implementation does not support data error correction for quality of service (QoS), such as cyclic redundancy code (CRC) calculation to detect errors in data communication such as presented by GEXSPidergon NoC [10]. The QoS in this level is certainly an interesting topic for further implementation of the XHiNoC.

Acknowledgments

The authors gratefully acknowledge *Deutscher Akademischer Austausch Dienst* DAAD, German Academic Exchange Service for awarding F. A. Samman with scholarship pursuing doctoral degree at Darmstadt University of Technology in Germany, as well as the comments and suggestions made by the reviewers.

References

[1] "The International Technology Roadmap for Semiconductors," Design Technology Roadmap, Update 2006, http://www.itrs.net.

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

[3] A. Jantsch and H. Tenhunen, *Networks on Chip*, Kluwer Academic Publishers, Hingham, Mass, USA, 2003.

[4] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *Proceedings*

of the Conference on Design, Automation and Test in Europe (DATE '04), vol. 2, pp. 890–895, Paris, France, February 2004.

[5] D. Wiklund and D. Liu, "SoCBUS: switched network on chip for hard real time embedded systems," in *Proceedings of 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS '03)*, p. 8, Nice, France, April 2003.

[6] M. B. Taylor, J. Kim, J. Miller, et al., "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

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

[8] M. K.-F. Schäfer, T. Hollstein, H. Zimmer, and M. Glesner, "Deadlock-free routing and component placement for irregular mesh-based networks-on-chip," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '05)*, pp. 238–245, San Jose, Calif, USA, November 2005.

[9] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, no. 5, pp. 36–45, 2002.

[10] M. Zid, A. Zitouni, A. Baganne, and R. Tourki, "New generic GALS NoC architecture with multiple QoS," in *Proceedings of IEEE International Conference on Design and Test of Integrated Systems in Nanoscale Technology (DTIS '06)*, pp. 345–349, Gammarth, Tunisia, September 2006.

[11] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnection," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '00)*, pp. 250–256, Paris, France, March 2000.

[12] I. M. Panades, A. Greiner, and A. Sheibanyrad, "A low cost network-on-chip with guaranteed service well suited to the GALS approach," in *Proceedings of the 1st International Conference on Nano-Networks and Workshops (NanoNet '06)*, pp. 1–5, Lausanne, Switzerland, September 2006.

[13] T. A. Bartic, J.-Y. Mignolet, V. Nollet, et al., "Topology adaptive network-on-chip design and implementation," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 4, pp. 467–472, 2005.

[14] C. Hilton and B. Nelson, "PNoC: a flexible circuit-switched NoC for FPGA-based systems," *IEE Proceedings: Computers and Digital Techniques*, vol. 153, no. 3, pp. 181–188, 2006.

[15] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, pp. 261–272, 2005.

[16] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A design methodology for application-specific networks-on-chip," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263–280, 2006.

[17] J. Bainbridge and S. Furber, "Chain: a delay-insensitive chip area interconnect," *IEEE Micro*, vol. 22, no. 5, pp. 16–23, 2002.

[18] M. Amde, T. Felicijan, A. Efthymiou, D. Edwards, and L. Lavagno, "Asynchronous on-chip networks," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, pp. 273–283, 2005.

[19] I. Saastamoinen, D. Sigüenza-Tortosa, and J. Nurmi, "Interconnect IP node for future system-on-chip designs," in *Proceedings of the 1st IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)*, pp. 116–120, Christchurch, New Zealand, January 2002.

[20] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '05)*, pp. 54–63, New York, NY, USA, March 2005.

[21] T. Bjerregaard and J. Sparsø, "Implementation of guaranteed services in the MANGO clockless network-on-chip," *IEE Proceedings: Computers and Digital Techniques*, vol. 153, no. 4, pp. 217–229, 2006.

[22] F. A. Samman, T. Hollstein, and M. Glesner, "Multicast parallel pipeline router architecture for network-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*, pp. 1396–1401, Munich, Germany, March 2008.

[23] E. Rijpkema, K. Goossens, A. Rădulescu, et al., "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," *IEE Proceedings: Computers and Digital Techniques*, vol. 150, no. 5, pp. 294–302, 2003.

[24] S. Vassiliadis and I. Sourdis, "FLUX interconnection networks on demand," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 777–793, 2007.

[25] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings of the 19th International Symposium on Computer Architecture*, pp. 278–287, Gold Coast, Australia, May 1992.

[26] C. J. Glass and L. M. Ni, "Adaptive routing in mesh-connected networks," in *Proceedings of the 12th International Conference on Distributed Computing Systems (ICDCS '92)*, pp. 12–19, Yokohama, Japan, June 1992.

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

[28] M. A. Al Faruque, T. Ebi, and J. Henkel, "Run-time adaptive on-chip communication scheme," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 26–31, San Jose, Calif, USA, November 2007.