# New Theory for Deadlock-Free Multicast Routing in Wormhole-Switched Virtual-Channelless Networks-on-Chip

Faizal Arya Samman, *Member, IEEE*, Thomas Hollstein, *Member, IEEE*, and Manfred Glesner, *Fellow, IEEE*

**Abstract**—A new theory for deadlock-free multicast routing especially used for on-chip interconnection network (NoC) is presented in this paper. The NoC router hardware solution that enables the deadlock-free multicast routing without utilizing virtual channels is introduced formally. The special characteristic of the NoC is that, wormhole packets can cut-through at flit-level and can be interleaved in the same channel with other flits of different packets by multiplexing it using a rotating flit-by-flit arbitration. The routing paths of each flit can be guaranteed correct because flits belonging to the same packet are labeled with the same local Id-tag on every communication channel. Hence, multicast deadlock problem can be solved at each router by further applying a hold-release tagging mechanism to control and manage conflicting multicast requests.

**Index Terms**—Network-on-chip, tree-based and multipath-based multicast routing, Id-tag-based wormhole packet switching, runtime adaptive routing and scheduling.

## 1 Introduction

Efficient routing is a very important aspect to optimize the performance and communication energy of data transports in the NoC-based multicore processor systems. Historically, the first generation multicomputers supported only unicast communication (a single PE sends a message to a single PE unit). Collective communication services have been nowadays implemented in the recent multicomputers. The collective communication services can be in forms of *multicast* (the same message is sent from a source node to an arbitrary number of destination nodes), *scatter* (different messages are sent from a source node to an arbitrary number of destination nodes), and *broadcast* (the same message is sent from a source node to all nodes in the network). The implementation of the collective communication services will not only simplify the programming model in software-layer but also will perform efficient routing in terms of the latency and energy of the data transmission.

Multicast service has been intensively used in large-scale multiprocessor systems and is a standard service of some data parallel computer languages. Data parallel languages such as *Fortran D* [1], *Distributed Fortran 90* [2], and *High-Performance Fortran* [3] have included multicast data transport routines for users to develop parallel computation and algorithms. In the single-program multiple-data (SPMD) programming model, multicast communication is of benefit, where the same program is executed on different processors with different data, and several data are processed in parallel. In a data parallel programming model, a variety of process control operations and global data movement such as *reduction*, *replication*, *permutation*, *segmented scan*, and *barrier synchronization* requires the collective communication models.

In a distributed shared memory paradigm, multicast services maybe used to efficiently support shared data invalidation and updating. Some libraries commonly used to develop parallel computer programs based on message passing programming model and shared memory programming model such as *Message Passing Interface* (MPI) [4], [5] and *Parallel Virtual Machine* (PVM) [5], [6] provide also multicast data transport routines and procedures to design explicitly parallel computation and communications. Both MPI and PVM libraries are available for C and Fortran Computer Languages. Numerous parallel algorithms, e.g., parallel search and parallel graph algorithms, have taken also the advantages of the multicast service [7]. By using software implementation, a multicast message can be sent to the network by replicating separate copies of the message from source to every destination node (unicast-based multicast delivery). However, this approach leads to inefficient transmission time and workload energy.

In order to achieve the effectiveness of the multicast data communication, the multicast service must be implemented not only in the higher protocol layers (software layers) but also in the lower layers (hardware layers). In the NoC

- *F.A. Samman is with the FB Elektrotechnik und Informationstechnik, Research Group on Microelectronic Systems, Technische Universitäẗ Darmstadt, Merckstr. 25, 64283 Darmstadt, and the Fraunhofer Institut LBF, LOEWE-Zentrum AdRIA (Adaptronik-Research, Innovation, Application), Bartningstr. 53, 64289 Darmstadt, Germany, and also with the Deparment of Electrical Engineering, Hasanuddin University at Makassar, Indonesia. E-mail: faizal.samman@mes.tu-darmstadt.de, faizal.samman@loewe-adria.de.*
- *T. Hollstein is with Department of Computer Engineering, Dependable Embedded Systems Group, Tallinn University of Technology, Estonia. E-mail: thomas.hollstein@ies.tu-darmstadt.de.*
- *M. Glesner is with FB Elektrotechnik und Informationstechnik, Research Group on Microelectronic Systems, Technische Universitaät Darmstadt, Merckstr. 25, 64283 Darmstadt, Germany. E-mail: manfred.glesner@mes.tu-darmstadt.de.*

*Manuscript received 2 July 2009; revised 7 Jan. 2010; accepted 3 Apr. 2010; published online 1 June 2010.*
*Recommended for acceptance by J. Flich.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-07-0297.*
*Digital Object Identifier no. 10.1109/TPDS.2010.120.*

context, these layers refer to network interface layer (transport layer) and on-chip routing layer (network, routing, and physical layers). The main problem of the multicast service implementation in the on-chip router is multicast dependencies between packets in the router that can lead to multicast deadlock configuration.

The remaining sections of the paper is organized as follows: Section 2 presents related work and our motivations to introduce the new theory. Section 3 defines the main problem with some general definitions. Section 4 proposes the technical solution and formalism of the hardware-based solution. The formal descriptions of some definitions, lemmas followed by the proof of the new theory are shown in Section 5. Section 6 finally presents the concluding remarks about the new theory.

## 2 Related Works and Contributions

Some theories and methodologies to perform a deadlock-free multicast routing have been presented in [7], [8], [9], [10], [11], and [12]. The works in [7], [8], [9] specially present the theory for deadlock-free *path-based multicast routing*, while the works in [10], [11], and [12] present the theory for deadlock-free *tree-based multicast routing*. However, the multicast theory and routing protocol presented in the aforementioned works are not dedicated for NoCs.

There have been some other works that have introduced multicast routing services dedicated for NoCs. The work in [13], for example, presents a *Multicast Router Rotary* ($MRR$). The multicast routing algorithm in the MRR can be classified into a distributed routing method. The multicast contention in MRR is solved by implementing two single-direction internal rings in the switch, one in clockwise direction and the other one in counterclockwise direction. Without careful data-flow rule, a dangerous permanent deadlock can occur especially when packets come from all different input ports, and each of them requests simultaneously all output ports. The proposed data-flow rule in the MRR must even allow misrouting to avoid deadlock in a case that a packet cannot find a free output port. In any circumstance, misrouting can increase data communication energy due to the overhead misrouting traffics, and can lead to livelock situation. The work in [13] has not yet addressed this livelock issue. Moreover, additional 10 internal buffers (5 for each ring) in the MRR will increase the area overhead of the router.

The work in [14] presents a *broadcast-multicast-enabled Logic-based Distributed Routing* ($bLBDR$). Another routing approach called *Recursive Partitioning Multicast* ($RPM$) method is also presented in [15]. The bLBDR and RPM methods need global network view and preprocessing algorithm for network partitioning. The bLBDR uses configuration bits at each router and broadcast/multicast operations are computed at every router. When using wormhole switching method, concurrent broadcast using the bLBDR can lead to deadlock. Since the bLBDR proposes a tree-based multicast routing inside region, our contribution in this paper can be a complementary solution for the deadlock problem. In the RPM method, a routing decision is made based on the current network partitioning that has been previously computed recursively, and the multicast deadlock problem is solved by using virtual channels. The whole network is divided into at most eight subnets by the source node. The objective of partitioning the network is to minimize packet replication time.

A *Virtual Circuit Tree Multicasting* ($VCTM$) method is presented in [16]. In the VCTM method, a setup packet must be sent in the network to configure a switched tree-based multicast virtual circuit. Like the RPM method [15], the VCTM solves the multicast deadlock problem by using virtual channels (VCs). The main critics of the use of VCs in the NoC context are prohibitive area cost in terms of buffering and/or potential slower speed of router cycle time. VCs will increase total buffer counts and result in power consumption that would exceed the target constraint for an embedded application [19]. The work in [17] has found that additional VCs do not increase router cycle time because the router complexity slows down the router working period. VCs can increase delay in router's critical path due to extra arbitrations, thus it potentially affects the cycle time or pipeline depth of the router [20]. The same result is presented in [18], in which additional arbitration and multiplexing circuits for VCs on physical channels introduce delay into the critical path in implementing alternate routing algorithms.

The MRR [13], bLBDR [14], and VCTM [16] rely mainly in virtual-cut-through networks to avoid multicast/broadcast deadlocks. The RPM [15] uses wormhole switching but it uses virtual channels leading to higher area overhead. The proposed solution in this paper is novel since it targets wormhole-switching network without using virtual channels. This paper will present a new theory to solve the multicast dependency problem suitable for wormhole-switched NoCs without virtual channels, where the routing algorithms used to route unicast and multicast packets are the same, resulting in a very efficient routing function gate-level implementation. The theory is practically applicable and implementable directly in NoC routing hardware layer using path-based and tree-based multicast routing. Runtime local/distributed routing is used as the basis solution rather than derive specific algorithms executed in software-level (source routing). Hence, the solution based on our new theory can be performed at runtime during application execution time. The new theory must be supported by a specific hardware infrastructure to back up the effectiveness of the new theory. The formal descriptions of the hardware base will be also described in this paper.

## 3 Problem Definition

**Definition 3.1 (Network-on-Chip).** *A network-on-chip (NoC) can be represented as a graph $G(\Re, \Lambda)$, where $\Lambda$ is represented as a set of edges (communication links) and $\Re$ is represented as a set of vertices (router nodes).*

**Definition 3.2 (Router).** *NoC consisting of $N_{node}$ number of node will have a set of NoC Router $\Re = \{R_1, R_2, \ldots, R_{N_{node}}\}$ or $R_c \in \Re | c = \{1, 2, \ldots, N_{node}\}$.*

**Definition 3.3 (Communication Link).** *Communication link $L_{i,j} \in \Lambda$ is a communication link connecting router node $R_i$ and $R_j$ where $R_i, R_j \in \Re$, and $i, j = \{1, 2, \ldots, N_{node}\}$.*

The number of link components in the set $\Re$ depends on network-on-chip topology. We can describe that $\neg\forall i, j$ such

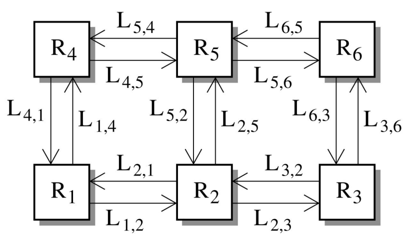

Fig. 1. A NoC in a 2D $3 \times 2$ mesh network graph.

that $L_{i,j}$ exists. Fig. 1 shows an example of a NoC in $3 \times 2$ mesh topology with full duplex links connection. Based on Definition 3.1, we can see that the router set is $\Re = \{R_1, R_2, R_3, R_4, R_5, R_6\}$, while the set of the communication resources is defined as $\Lambda = \{\Lambda_{horz}, \Lambda_{vert}\}$, where $\Lambda_{horz} = \{L_{1,2}, L_{2,1}, L_{2,3}, L_{3,2}, L_{4,5}, L_{5,4}, L_{5,6}, L_{6,5}\}$ and $\Lambda_{vert} = \{L_{4,1}, L_{1,4}, L_{2,5}, L_{5,2}, L_{3,6}, L_{6,3}\}$.

**Definition 3.4 (Router IO Port).** *For $N_{inp}$ number of input ports and $N_{outp}$ number of output ports of a router $R \in \Re$, we can describe the set of input ports as $\rho_I = \{I_1, I_2 \ldots, I_{N_{inp}}\}$ and the set of output ports as $\rho_O = \{O_1, O_2 \ldots, O_{N_{outp}}\}$. Hence, if we define $\Phi = \{1, 2, \ldots, N_{inp}\}$ and $\varphi = \{1, 2, \ldots, N_{outp}\}$, then we can define an input port of router as $I_n \in \rho_I | n \in \Phi$, and an output port of a router as $O_m \in \rho_O | m \in \varphi$. We can further define that $\forall n$: $I_n = n$ and $\forall m$: $O_m = m$.*

Multicast deadlock configuration is a situation in which multicast packets, which are switched in the network with wormhole switching technique, cannot move further due to multicast dependency occurs in some NoC routers. The multicast dependency occurs because two or more multicast packets are competing each other to access the same output ports in any NoC router, while in other NoC routers, the same situation occurs, i.e., the same competing multicast packets compete also to acquire the same output ports. In order to understand a better insight about the multicast deadlock configuration, an example of the multicast deadlock configurations are presented in Fig. 2. The deadlock configuration when using tree-based multicast routing with wormhole switching method is shown in Fig. 2a. In node (2,2), message $A$ cannot move further because the East and West output ports are acquired by message $B$. Meanwhile, message $B$ cannot advance in node (2,1), because the East and West output ports are acquired by message $A$. The multicast dependencies of the contenting multicast packets in many network nodes can lead to a deadlock configuration.

In a path-based multicasting, a source node arranges the ordered list of headers containing destination address. When a message is injected to the network, it will be routed to a destination node according to the address attached in the leading header flit. When the message arrives the destination node, the leading header flit is removed. Hence, the next header will be the leading flit and guides the message into the next destination. The path-based multicasting is a mechanism to avoid branching in intermediate nodes. Two branches of the paths are formed only in destination nodes, i.e., one to Local port and one to another port. In the source node, the number of branches can be maximum two for dual-path and more than two for the multipath-based multicasting. Deadlock configuration can occur when a multicast router does tree branch (even by using dual/multipath multicast) as shown in Fig. 2b.

## 4 Technical Solution and Formal Description

### 4.1 Multicast Contention Control and Management

Deadlock configurations because of multicast contentions can occur not only in tree-based but also in multipath-based multicast routing. The deadlock occurs in an intermediate node when one or more outgoing links are simultaneously requested by the same multicast packet. We propose a novel scheduling method called *Hold-Release Tagging Mechanism for fair multicast scheduling policy* to manage the multicast contention for a NoC without using virtual channels. The methodology is suitable for NoC router supporting simultaneous parallel crossbar interconnection, where a routing unit and an arbitration are distributed on every input port and every output port, respectively.

The philosophy of the *Hold-Release Tagging Mechanism* is as follows: "If a multicast flits from an input port $n$ has an $N_{s,n}^{req}$ number of requests at any instant of time $t_s$, then each single request to an output port $m$ can be forwarded from

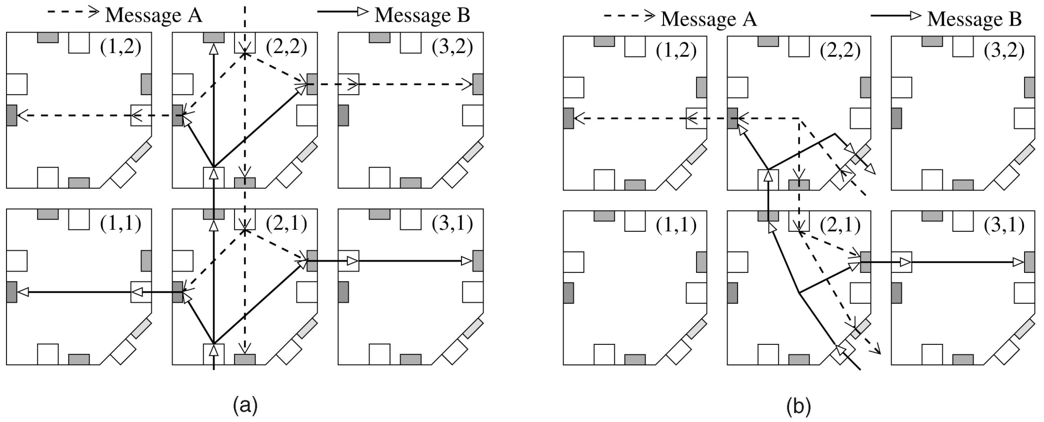

Fig. 2. Multicast deadlock configurations when using tree-based and path-based multicast routing in mesh networks. (a) Deadlock in tree-based multicast routing. (b) Branching in dual/multipath multicasting.

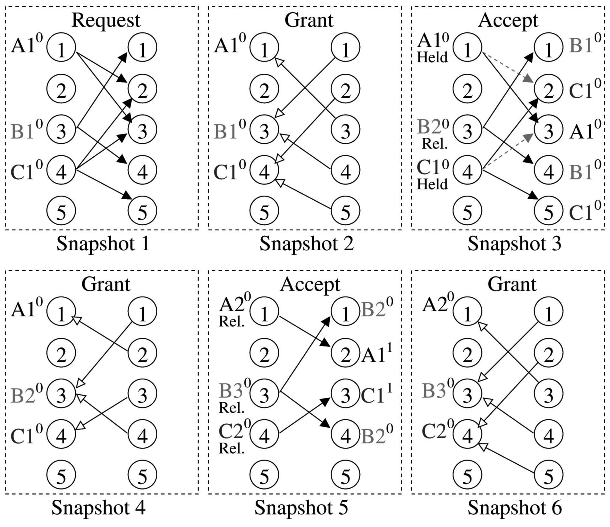

Fig. 3. Hold-Release tagging mechanism for a fair multicast flit arbitration and scheduling policy.

the input port $n$ to the output port $m$ in the next time stage only if it receives a grant by an arbitration unit at input port $n$, while the other requests must be held in the input port if it is not granted by their requested output ports. In each next time stage, a single request, which has been granted before, must be reset to prevent improper flit replication. If all requests have been granted, then the multicast flit can be released from the queue in input port $n$."

In order to support such mechanism, the flits of the wormhole packets must be able to be interleaved/multiplexed flit-by-flit to share the same link with other flits belonging to different wormhole packets (cut-through at flit-level). The "wormhole cut-through switching" is implementable by attaching a local ID-tag on each flit of packets, where flits belonging to the same packets will have the same local ID-tag on every local communication channel [24], [25], [26]. Furthermore, an arbitration unit distributed at every output port $m$ must rotate (circulate) its selection among requests at instant time $t_s$.

Fig. 3 presents successive snapshots on how the proposed method solves the deadlock problem due to the multicast dependency. In the Snapshots 2, 4, and 6, we can see how the selection of each arbiter unit at every output port is rotated among requests from input ports. In Snapshot 3, we can see how requests, which are not granted in the previous grant step (Snapshot 2), are not forwarded to the output ports (depicted in dashed lines). Snapshot 5 shows how requests that have been granted are reset to prevent improper flit replication. In order to guarantee the correct routing path and proper replication in the hold-release tagging mechanism, both issues will be discussed formally in Sections 5.1 and 5.2, respectively.

## 4.2 Specific Packet Format

Fig. 4 presents the generic specific packet format that should be used to perform a deadlock-free multicast routing and to enable the hold-release tagging mechanism described in Section 4.1. A message or a streaming datum is divided into flits. The total bit-width of each flit of the message or

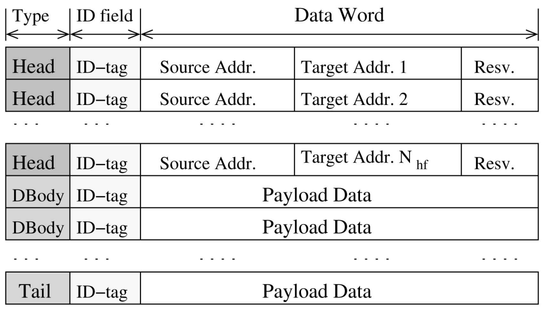
Type
ID field
Data Word
Head | ID–tag | Source Addr. | Target Addr. 1 | Resv.
Head | ID–tag | Source Addr. | Target Addr. 2 | Resv.
Head | ID–tag | Source Addr. | Target Addr. N hf | Resv.
DBody | ID–tag | Payload Data
DBody | ID–tag | Payload Data
Tail | ID–tag | Payload Data

Fig. 4. Specific packet format.

streaming data is $B_{total} = B_{type} + B_{tag} + B_{word}$, where $B_{type}$ is the bit-width of the flit-type field, $B_{tag}$ is the bit-width of the id-tag field, and $B_{word}$ is the bit-width of the data word. Formally, we will give formal definition for unicast and multicast message or streaming data.

Unicast message/stream is single packet consisting of one header flit, databody flits, and one tail flit, while multicast message/stream consists of more than one header flits. Even if the size of the message/stream is extremely large, it has only one tail flit. In other words, a message or data stream is assembled in a single packet.

A number of header flit $N_{hf}$ represents the number of the multicast destination $N_{dest}$ ($N_{hf} = N_{dest}$). The header flit contains information of the source address from which node the message is injected and the target address to which nodes the message will be sent. If $N_{df}$ number of data flits will be sent to $N_{dest}$ number of multicast target, then the total number of flits injected to the NoC is $N_{Flit} = N_{df} + N_{hf}$.

**Definition 4.1 (Data Flit).** *A data flit coming from input port $n$ is represented as $F_n(type, Id)$ that consists of a data word with additional flit-type field (type) and local Id-tag field ($Id$). Each flit will always bring a data word together with its type and its local (see Fig. 4).*

**Definition 4.2 (Flit type).** *The type field represents the type of each flit. The flit-type can be a header, a databody, a tail flit, or a response flit ($type \in \varepsilon_{type} | \varepsilon_{type} = \{header, databody, tail, response\}$).*

**Definition 4.3 (Flit Id-tag).** *Id-tag field present on each flit is a local label (Id-tag) to indicate and differentiate the flit from different flits. Flits belonging to the same message or streaming data will always have the same local Id-tag on each communication link $L_{i,j} \in \Lambda$. The value of the local Id-tag is defined as $Id \in \Gamma | \Gamma = \{0, 1, 2, \ldots, N_{slot} - 1\}$ where $N_{slot}$ is number of available Id slot on communication link $L_{i,j} \in \Lambda$. See also later the definition of the local Id slot in Definition 4.4.*

**Definition 4.4 (Local Id Slots).** *Each communication link $L_{i,j} \in \Lambda$ has $N_{slot}$ number of available local Id slots, which is defined as a set $\Omega_{i,j} \subseteqq \Gamma$ (see Definition 4.3). If we assume that all communication links has equal set of available Id slots then the term $\Omega_{i,j} \subseteqq \Omega$. We define a single local Id slot $k \in \Omega$, where $k = N_{slot} - 1$ is reserved for flow-control purpose, and the usable Id slots are $\forall k \in \Omega \cap k \neq N_{slot} - 1$.*

## 4.3 Specific Routing Organization

**Definition 4.5 (Routing Engine).** *Routing Engine (RE) is a component to make a routing direction $r_{dir} \in D$ where*

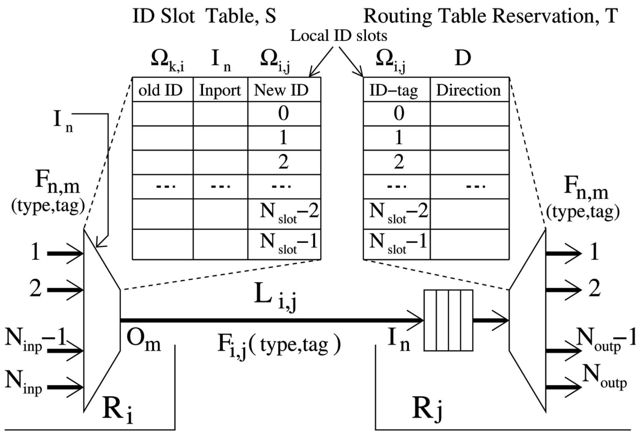

Fig. 5. Local Id slots per link.

*$D = \{1, 2, \ldots, N_{outp}\}$. For $N_{inp}$ number of router input ports, then the number of RE per router is $N_{RE} = N_{inp}$. We can describe that the RE at input port $I_n \in \rho_I$ as $E_n, n \in \rho_I$. The $E_n$ provides a routing function $f_{RE}$ giving routing direction D. We can defined $f_{RE}$: $r_{dir} = f_{RE}(type, Id, A_{dest})$, where $A_{dest}$ is the destination address field present in the header flit.*

The RE ($E_n$) at input port $n \in \Phi$ consists of combination of a *routing state machine* $M$ and *routing reservation table* $T$. Accordingly, there will be pairs of $(M_n, T_n)|\forall n \in \Phi$. Both components are allocated at each input port. Operation in the Routing Engine depends on the type of the data flit $F_n(type, Id)$. The type of flit will also determine which component ($M$ or $T$) that will give a routing direction.

**Definition 4.6 (Routing State Machine).** *The routing state machine (RSM) provides a routing function $f_{RSM}$ where the output of the function depends on the destination address appear on the target address field $A_{dest}$ of a header flit. If a header flit is detected by the routing state machine, then the routing direction of the flit is computed with routing function $f_{RSM}$: $f_{RSM}(A_{dest}) \Rightarrow r_{dir}$, where $A_{dest}$ is the destination address present on the header flit and $r_{dir} \in D$.*

**Definition 4.7 (Routing Reservation Table).** *A Routing Reservation Table (RRT) of the RE unit at an input port is defined as*

$$T(k|k \in \Omega) = r_{dir} \in D = \{1, 2, 3, \ldots, N_{outp}\}, \quad (1)$$

*or $T(k) \in D|k \in \Omega$. Definition of $D$ can be found in Definition 4.6. So, we can define that $\forall k \in \Omega$, the value of the Routing Table $T(k)$ is a routing direction $r_{dir} \in D$.*

The array structure of the RRT $T$ can be seen in Fig. 5. As shown in Algorithm 1, when a header flit $F_{n,m}(header, Id)$ is coming from an input port $n$, a routing direction $r_{dir}$ is computed by the RSM, and it is concurrently written (copied) in the slot number $k$ in the RRT, where $k = Id$ (equal to the Id-tag of a header flit) such that $T(Id) = r_{dir}$. When a databody $F_{n,m}(header, Id)$ or tail flit $F_{n,m}(header, Id)$ is coming to the input port $n$, then the $r_{dir}$ is taken from the slot number $k = Id$ in the RRT. The operation $r_{dir} \cup T(Id)$ in the algorithm is the union operation between the current content of the $T(Id)$ in the slot number $Id$ and the current value of the routing direction $r_{dir}$.

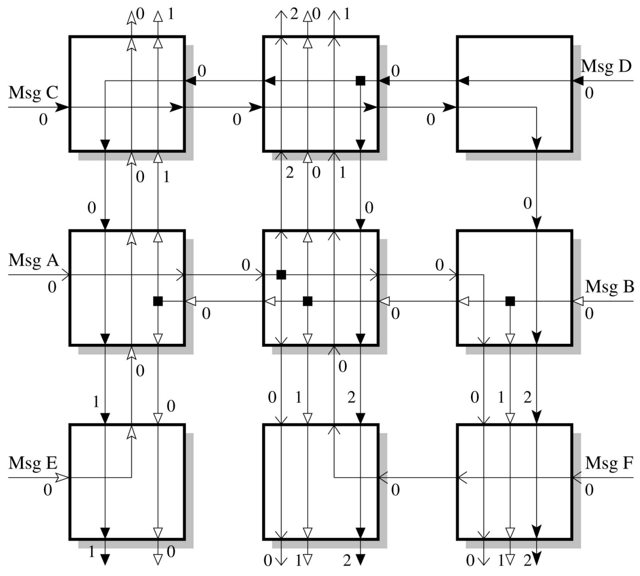

Fig. 6. Flexible runtime interconnect scheduling and switching based on locally variable/dynamic Id-tagging.

**Algorithm 1.** Runtime Id-based Routing Organization
Read Data Flit from Queue : $F_n(type, Id)$

```
1: $F_{type} \Leftarrow type$
2: $A_{dest}$ is obtained from Header flits
3: if $F_{type}$ is header or response then
4:     $r_{dir} \Leftarrow f_{RSM}(A_{dest})$; $T(Id) \Leftarrow r_{dir} \cup T(Id)$
5: else if $F_{type}$ is databody then
6:     $r_{dir} \Leftarrow T(Id)$
7: else if $F_{type}$ is tail then
8:     $r_{dir} \Leftarrow T(Id)$; $T(Id) \Leftarrow \emptyset$
9: end if
```

## 4.4 Specific Switching and Arbitration Method

The hardware solution to overcome the multicast deadlock configuration is realized by organizing locally the Id-tag of each multicast message on each communication link. The local Id-tag of each branch of the multicast tree is variable and organized by an Id-tag management unit. Fig. 6 shows how the local Id-tags of all multicast tree branches are dynamically assigned, where a different message must be allocated to different Id slot in the same one-directional link. The number in each one-directional link is the Id slot, to which the message is allocated. Every communication link is shared fairly by applying a rotating flit-by-flit arbitration as defined in Definition 4.11. Section 4.5 will formally describe the other specific hardware requirement to solve the multicast problem.

**Definition 4.8 (Time-Varying Binary Request).** *A time-varying input-output request from an input port $n \in \Phi$ to an output port $m \in \varphi$ is defined as $r_{n,m}(t)$: $\forall n, m|r_{n,m}(t) \in \{0, 1\}$. Hence, the* time-varying input $n$ binary request *can be defined as array of binary or $r_{n,m=1 \text{ to } N_{outp}}(t)$ or $r_{n,m=1:N_{outp}}(t)$ or $r_{n,1:N_{outp}}(t)$, and the* time-varying output $m$ *binary* request *is defined also as $r_{n=1 \text{ to } N_{inp},m}(t)$ or $r_{n=1:N_{inp},m}(t)$ or $r_{1:N_{inp},m}(t)$.*

**Definition 4.9 (Set of Requests from an Input Port).** *A set of requests from an input port $n$ to output ports is defined as*

*$\varphi_n^{req}$, thus the number of set members $N_{s,n}^{req}$ is defined as the number of requests from an input port $n$ to output ports at time stage $t_s$. Further, we can define the definitions in the following equation:*

$$\begin{aligned}
&\varphi_n^{req} \subset \varphi \Leftrightarrow N_{s,n}^{req} = \sum_{m=1}^{N_{outp}} r_{n,m}(t_s) < N_{outp} \\
&\varphi_n^{req} \subseteq \varphi \Leftrightarrow N_{s,n}^{req} = \sum_{m=1}^{N_{outp}} r_{n,m}(t_s) = N_{outp} \\
&\varphi_n^{req} = \emptyset \Leftrightarrow N_{s,n}^{req} = \sum_{m=1}^{N_{outp}} r_{n,m}(t_s) = 0 \\
&h \in \varphi_n^{req} \Leftrightarrow r_{n,h}(t) = 1.
\end{aligned} \tag{2}$$

*For example, if $r_{n,1:N_{outp}}(t_s) = [1 \quad 0 \quad 1 \quad 1 \quad 0]$ or $r_{n,1}(t_s) = 1$, $r_{n,2}(t_s) = 0$, $r_{n,3}(t_s) = 1$, $r_{n,4}(t_s) = 1$ and $r_{n,5}(t_s) = 0$, then $N_{s,n}^{req} = 3$ and $\varphi_n^{req} = \{1, 3, 4\}$.*

**Definition 4.10 (Set of Requests to an Output Port).** *A set of requests to an output port $m$ from input ports is defined as $\Phi_n^{req}$, thus the number of set members $N_{s,m}^{req}$ is defined as the number of requests to an output port $m$ from input ports at time stage $t_s$. Further, we can define the definitions in the following equation:*

$$\begin{aligned}
&\Phi_m^{req} \subset \Phi \Leftrightarrow N_{s,m}^{req} = \sum_{n=1}^{N_{inp}} r_{n,m}(t_s) < N_{inp} \\
&\Phi_m^{req} \subseteq \Phi \Leftrightarrow N_{s,m}^{req} = \sum_{n=1}^{N_{inp}} r_{n,m}(t_s) = N_{inp} \\
&\Phi_m^{req} = \emptyset \Leftrightarrow N_{s,m}^{req} = \sum_{n=1}^{N_{inp}} r_{n,m}(t_s) = 0 \\
&l \in \Phi_m^{req} \Leftrightarrow r_{l,m}(t) = 1.
\end{aligned} \tag{3}$$

*For example, if $r_{1:N_{inp},m}(t_s) = [0 \quad 1 \quad 1 \quad 0 \quad 1]$ or $r_{1,m}(t_s) = 0$, $r_{2,m}(t_s) = 1$, $r_{3,m}(t_s) = 1$, $r_{4,m}(t_s) = 0$ and $r_{5,m}(t_s) = 1$, then $N_{s,n}^{req} = 3$ and $\Phi_m^{req} = \{2, 3, 5\}$.*

**Definition 4.11 (Rotating Flit-by-Flit Arbitration).** *A rotating flit-by-flit arbitration is an arbitration strategy at every output port $m$ that circulates its selection among input ports $l \in \Phi_m^{req}$ according to Definition 4.10 and (3) in flit-by-flit manner.*

Algorithm 2 shows the rotating/circulating flit-by-flit arbitration at every output port $m$. At initial time (no request at all), the arbiter will first serve the first coming flit. When many flits need arbitration service at the same cycle, the arbiter will rotate selection like round-robin arbiter, but the arbitration is done in flit-by-flit and active-port-by-active-port manner. Active port refers to an input port having request for routing. It means that the arbiter circulates selection but will not select input ports having no data flit.

**Algorithm 2.** Rotating Flit-by-Flit Arbitration

```
Binary Request  :  $r_n \in \{0, 1\} : n = 1, 2, \ldots, N_{inp}$
Initial Value   :  $Rot_{Start} = N_{inp}$, $Rot_{Stop} = 1$

1: while $\exists n : r_n \neq 0$ & the next queue is $\neg Full$ do
2:   for $n = Rot_{Start}$ down to $n = Rot_{Stop}$ do
3:     if $r_n = 1$ then
4:       $select \Leftarrow n$; $Rot_{Start} = n - 1$; $Rot_{Stop} = n$
5:     end if
6:   end for
7: end while
8: Result: $select \Rightarrow I_n$
```

**Definition 4.12 (Rotating Arbitration Time).** *Input port selection in every output port $m$ will be recirculated in every $T_{s,m} + 1$, where $T_{s,m} = N_{s,m}^{req}$ is a Rotating Arbitration Time.*

**Definition 4.13 (Binary Output Acknowledgment).** *We have defined the time-varying output $m$ binary routing request in Definition 4.8. Thus, we can now define the Request-*Dependent Binary Output Acknowledgment *as $a_{1:N_{inp},m}(t)$. For $\forall n | a_{n,m}(t) \in \{0, 1\}$ where only one element of $a_{1:N_{inp},m}(t)$ can be set to '1' because of natural hardware constraint of an arbiter unit. When $N_{s,m}^{req} > 1$, then physically it means that there is contention between $N_{s,m}^{req}$ number of flits between input ports in the set $\Phi_m^{req}$ to access the same output port $m$. For example, if the binary request $r_{1:5,m}(t_s) = [0 \quad 1 \quad 0 \quad 1 \quad 1]^T$, then according to Definition 4.12 and (3), $T_{s,m} = 3$. The set of possible order of the rotating arbitration at output port $m$ is $a_{1:5,m}(1) = [0 \quad 0 \quad 0 \quad 0 \quad 1]^T$, $a_{1:5,m}(2) = [0 \quad 0 \quad 0 \quad 1 \quad 0]^T$, and $a_{1:5,m}(3) = [0 \quad 1 \quad 0 \quad 0 \quad 0]^T$. According to Definition 4.10, $\Phi_m^{req} = \{2, 4, 5\}$. We can see that $r_{1:5,m}(t_s) = \bigcup_{t=1}^{t=3} a_{1:5,m}(t)$.*

## 4.5 Local Id-Tag Management

For each communication resource (link/channel) $L_{i,j}$ connecting an outport port $O_m \in \rho_O$ of an on-chip router $R_i$ with an input port $I_n \in \rho_I$ of an adjacent (neighbor) router $R_j$, then an amount of local identity (Id) slots $N_{slot}^{i,j}$ is implemented on the communication link $L_{i,j} \in \Lambda$.

**Definition 4.14 (ID Slot Table).** *ID Slot Table State is defined as a set of Slot State $S = \{S^0, S^1, S^2, \ldots, S^{N_{slot}-1}\}$ and $\forall k \in \Omega : S^k \in \{true, false\}$ (See Definition 4.4). If $S^k = true$, it means that the Id Slot state is "free," else if $S^k = false$, then the Id Slot is being "used" by any message. We can define the ID Slot Table as*

$$\forall k \in \Omega : S(k) = (Id_{old}, F_{from}) \in (\Omega, \Phi). \tag{4}$$

*The set $\Phi = \{1, 2, 3, \ldots, N_{inp}\}$ is in accordance with Defnition 3.4. We define $F_{from}$ as the selected flit from any input port as the arbitration result.*

Fig. 5 can help to comprehend Definition 4.14 and describes the structure of the ID Slot Table $S$. The figure presents the ID Slot Table in output port of Router $R_i$ and the Routing Table in the input port of Router $R_j$. A communication link $L_{i,j}$ connecting the output and input port of the $R_i$ and $R_j$. For the sake of simplicity and for maintaining a design regularity, we assume in this paper that $N_{slot}^{ij} = N_{slot}$. Therefore, the number of available Id-tags on each communication link is uniform.

As shown in Algorithm 3, when a header flit $F_n(header, Id)$ with ID-tag $Id$ coming from input port $n$ is switched to an output port $m$ then an Id-tag update function $f_{IDM}$: $(Id_{old}) \mapsto Id_{new}$ is made, and a free Id slot is searched for the header. When a free Id slot $k$ is found, the input port $n$ and the Id-tag $Id$ are written in the slot number $k$ in the $S$, and the header uses this Id slot $k$ as its new Id-tag ($Id_{new} = k$).

When the header fails to find a free Id slot, it will be assigned to Id slot $N_{slot} - 1$. When a databody $F_n(databody, Id)$ or tail $F_n(tail, Id)$ flit having the same Id-tag with the previously routed header $F_n(header, Id)$ flit is switched to the output port $m$, then they will be assigned with the same new Id-tag $k$. The databody and tail flits will be dropped from the network if its header having the same Id-tag fails to reserve an Id Slot $k \in \Omega \cap k \neq N_{slot} - 1$ on a certain link. Like response flits, header flits having ID-tag $N_{slot} - 1$ will be always routed in the NoC with the ID-tag $N_{slot} - 1$, in which their paths can be guaranteed correct even when many $F_n(header, N_{slot} - 1)$ and $F_n(response, N_{slot} - 1)$ flits flow in the NoC, since these flits are only single-flit, in which destination address is attached in their address field and independent from the routing table contents.

**Algorithm 3.** Runtime Local Id-tag Update

| | | |
|---|---|---|
| Outgoing Data Flit | : | $F_n(type, Id)$ |
| Input Arbitration | : | $n = \{1, 2, \ldots, N_{inp}\}$ |
| $N_{freeId}$ | : | number of free Id slots |
| $S^{N_{slot}-1}$ | : | Slot reserved for control purpose |

```
1:  $Id_{old} \Leftarrow Id$
2:  if $F_{type}$ is $header$ then
3:      if $Id_{old} = N_{slot} - 1$ then
4:          $Id_{new} \Leftarrow N_{slot} - 1$
5:      else if $Id_{old} \neq N_{slot} - 1$ then
6:          for $k = 0$ to $k = N_{slot} - 2$ do
7:              if $\exists k : S^k$ is true then
8:                  $S(k) \Leftarrow (Id_{old}, F_{from})$
9:                  $S^k \Leftarrow false$ /* the Id Slot is used now */
10:                 $Id_{new} \Leftarrow k$; $N_{freeId} \Leftarrow N_{freeId} - 1$
11:             else
12:                 $Id_{new} \Leftarrow N_{slot} - 1$
13:             end if
14:         end for
15:     end if
16: else if $F_{type}$ is $databody$ then
17:     for $k = 0$ to $k = N_{slot} - 1$ do
18:         if $\exists k : k \neq N_{slot} - 1 : S(k) = (Id_{old}, F_{from})$ then
19:             $Id_{new} \Leftarrow k$
20:         else if $\nexists k : k \neq N_{slot} - 1 : S(k) = (Id_{old}, F_{from})$ then
21:             $Id_{new} \Leftarrow \emptyset$; The $Databody$ flit is dropped
22:         end if
23:     end for
24: else if $F_{type}$ is $tail$ then
25:     $N_{freeId} \Leftarrow N_{freeId} + 1$
26:     for $k = 0$ to $k = N_{slot} - 1$ do
27:         if $\exists k : k \neq N_{slot} - 1 : S(k) = (Id_{old}, F_{from})$ then
28:             $Id_{new} \Leftarrow k$; $S(k) \Leftarrow (\emptyset, \emptyset)$
29:             $S^k \Leftarrow true$ /* the Id Slot is now free */
30:         else if $\nexists k : k \neq N_{slot} - 1 : S(k) = (Id_{old}, F_{from})$
                then
31:             $Id_{new} \Leftarrow \emptyset$; The $Tail$ flit is dropped
32:         end if
33:     end for
34: else if $F_{type}$ is $response$ then
35:     $Id_{new} \Leftarrow N_{slot} - 1$
36: end if
37: $Id_{new} \Rightarrow Id$
```

### 4.6 Issue Related to Local ID Slots Scalability

The main issue related to the local ID-based method for flexible communication media share is a *runout of local ID problem*. Therefore, it is important to set the number of available Id slots on each link such that all considered traffics can be covered to flow on every link. In a certain NoC topology, there will be a fact that there are some links that are never be used by packets sent from some computing element cores to some destination cores. Let us take a case of a NoC in 2D $N \times M$ mesh topology, and let us also set the address and port names of each node as $(x, y, O_p)$, where $x$ is the horizontal address such that $0 \leq x \leq N - 1$, $y$ is the vertical address such that $0 \leq y \leq M - 1$, and $O_p \in \{E(East), N(North), W(West), S(South), L(Local)\}$. Thus, When each communication edge of the 2D $N \times M$ mesh architecture is implemented with full-duplex link, then the number of available local Id slots in the North, South, East and West port can be set to a minimum number, i.e., less than $N \times M$ number of local Id slots.

When a *minimal fully adaptive (MFA)* routing algorithm is used to route packets, then the minimum number of the available Id slots that can be set to each output port of every NoC router is shown in (5).

$$N^{MFA}_{x,y,O_p} = \begin{cases} M(x+1), & 0_p = E, & 0 \leq x < N-1, \\ N(y+1), & 0_p = N, & 0 \leq y < M-1, \\ M(N-x), & 0_p = W, & 0 < x \leq N-1, \\ N(M-y), & 0_p = S, & 0 < y \leq M-1, \\ NM-1, & 0_p = L, & \\ 0, & otherwise. & \end{cases} \quad (5)$$

When a *Static X-First (SXF)* routing algorithm is used to route packets, then the minimum number of the available Id slots that can be set to the East and West Port will be less, i.e., $N^{SXF}_{x,y,E} = x + 1$ when $0 \leq x < N - 1$, and $N^{SXF}_{x,y,W} = N - x$ when $0 < x \leq N - 1$. Equation (5) is derived with assumption that a router node will not send packets to itself. The minimum number of Id slots at the Local output port is set to $(NM - 1)$ to anticipate an *all-to-one* communication, a collective communication mode, where all nodes send message to one target node. By setting the number of Id slots at each output port according to (5), we can guarantee that the Id slot runout problem in the 2D $N \times M$ mesh architecture can be avoided. Therefore, packet dropping and retransmission protocol can be neglected because both mechanisms may end up in unfairness for some traffic flows. The $NM$ quantity represents the number of participating computing elements in the NoC. If the number of processing elements connected to some mesh nodes is larger than one, then the (5) must be reformulated to cover all possible traffics flowing through each outgoing port of the mesh nodes.

## 5 Formalism of the New Theory

This section presents several definitions and lemmas that will be used to induce the proof of the new theorem for the deadlock-free multicast routing.

### 5.1 Correctness of the Routing Path Establishment Based on Local Id-Tag Management

**Lemma 1.** *By organizing the Id-tag of each flit of packets using Algorithm 3 with local ID Slot Table defined in Definition 4.14, then we can guarantee that flits belonging to the same packet will* *always have the same local Id-tag $k \in \Omega$ on each communication link $L_{i,j} \in \Lambda$.*

**Proof of Lemma 1.** Based on Definitions 4.4 and 4.14, we can see that the local Id slot $k \in \Omega$ is indexed by using two variables, i.e., the Id-tag $Id \in \Omega$ of a message from an input link and from which port $n \in \Phi$ the message come. Further, we define $F_n(type, Id)$ as a flit from input port $n$ with local Id-tag $Id$.

If we make a *preassumption*, that every single Id slot $k$ is allocated for every flit belonging to the same message on every link $L_{i,j}$ connected to an input port $n$, then $(\forall v, w \in \Omega_{i,j}) \cap (v \neq w)$, a flit $F_n(type, v)$ will not belong to the same message with $F_n(type, w)$, where $\Omega_{i,j}$ is the set of local Id slots on the link $L_{i,j}$. $\forall p, q \in \Phi \cap p \neq q$, there is a probability that $F_p(type, v)$ and $F_q(type, w)$ have the same local Id-tag $(v = w)$, because the sets of local Id Slot on each communication link is the same. But certainly $F_p(type, v)$ and $F_q(type, w)$ are flits of different packets, although $v = w, \because p \neq q$. If every single Id slot $k$ is assigned to a new packet by identifying two parameters, i.e., from which port the packet come and its current old Id-tag, we can make sure that $(\forall x, y \in \Omega) \cap (x \neq y) \Rightarrow S(x) = (v, p) \neq S(y) = (w, q)$ according to (4), $\because \forall p, q \in \Phi : p \neq q$, or if $p = q$ (ports connected to the same link $L_{i,j}$) then $v \neq w, \forall v, w \in \Omega_{i,j}$ according to the *preassumption*. $x$ and $y$ is the new Id-tags for flit $F_p(type, v)$ and $F_q(type, w)$, respectively.

Therefore, by further applying a local ID-tag management described in Algorithm 3, we can make sure that each different message can be allocated to one Id slot $k \in \Omega$ and guarantee that flits belonging to the same packet can be assigned to the same local Id-tag $k \in \Omega$ on each communication link. Accordingly, the proof makes also the aforementioned *preassumption* to be a *valid assumption* continuously on every communication link. □

**Lemma 2.** *By using Id-based routing mechanism as described in Algorithm 1 and organizing the Id-tag of each flit of packets as described in Algorithm 3, then each flit belonging to the same packet can be routed to a correct routing direction $r_{dir} \in D$ although the flits are interleaved with other flits that belong to other different message by applying the rotating flit-by-flit arbitration described in Definition 4.11.*

**Proof of Lemma 2.** Based on the **Proof of Lemma** 1, if we have proved that different packets can be allocated to different local Id Slot such that flits belonging to the same packet will always have the same local Id-tag, then by implementing ID Slot Table at every outgoing port such that if $\exists S^k, S(k) | k \in \Omega$ on each communication link $L_{i,j} \in \Lambda$ connecting routing node $R_i, R_j \in \Re$, then we can also further implementing a Routing Table $T(k)$ (Definition 4.7) at every incoming port, which routes flits of packets based on their Id-tag $k$ in such a way that the interleaved different flits can be correctly routed into their correct paths. □

## 5.2 Correctness of the Proper Multicast Flit Replication Based on Hold/Release Tagging Mechanism

**Definition 5.1.** *A Routing Request Matrix $R(t)$ describes the requests of all incoming flits to access the output ports at time stage unit $t$. The elements of the routing request matrix $R(t)$ consist of the elements of the time-varying input $n$ binary request or the time-varying output $m$ binary request defined in Definition 4.8 such that*

$$R_{N_{inp}\times N_{outp}}(t) = \left(r_{n,1}(t), r_{n,2}(t), \ldots, r_{n,N_{outp}}(t)\right),\quad n = \{1, 2, \ldots, N_{inp}-1, N_{inp}\} = \{1 : N_{inp}\} \tag{6}$$

$$R_{N_{inp}\times N_{outp}}(t) = \begin{pmatrix} r_{1,m=1:N_{outp}}(t) \\ r_{2,m=1:N_{outp}}(t) \\ \cdots \\ r_{N_{inp},m=1:N_{outp}}(t) \end{pmatrix}. \tag{7}$$

*We can also define that $R(t) : r_{n,m}(t) \in \{0, 1\}$, where $n$ and $m$ represent the row and column coordinates of each matrix element. According to Definition 3.4, $n$ and $m$ are interpreted as the input and output port number of the router IO port, respectively. The value of the $r_{n,m}(t)$ are either 0 or 1. The element $r_{n,m}(t) = 1$ if there is a routing request from input port $n$ to output port $m$, else its value is $r_{n,m}(t) = 0$. For a unicast request, $\forall n : \sum_{m=1}^{N_{outp}} r_{n,m}(t) = 1$, and for multicast request $\forall n : 1 < \sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq N_{outp}$.*

*If $0 \leq \sum_{n=1}^{N_{inp}} r_{n,m}(t) \leq 1$, then there is no contention to access the output port $m$. Equation (8) (left side) shows an example of the $R(t)$ for the Snapshot 1 in Fig. 3 where the IO ports are represented as port numbers 1, 2, 3, 4, and 5, respectively.*

$$R(1) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; A(1) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{8}$$

**Definition 5.2.** *An* Arbitration Matrix *$A$ describes the grant signal from an arbiter unit to select one flit from the input port to access its requested output port at time stage $t$. The Arbitration Matrix and its array elements are defined as $A(t) : a_{n,m}(t) \in \{0, 1\}$. The form of the Arbitration Matrix $A(t)$ is strongly dependent on $R(t)$.*

$$A_{N_{inp}\times N_{outp}}(t) = \left(a_{n,1}(t), a_{n,2}(t), \ldots, a_{n,N_{outp}}(t)\right)\quad n = \{1, 2, \ldots, N_{inp}-1, N_{inp}\} = \{1 : N_{inp}\}. \tag{9}$$

*For example, if the output port $m = 2$ has two requests from input ports as shown in column 2 of matrix $R(t)$ in (8), i.e., $r_{n=1:5,m=2}(t) = [1\ \ 0\ \ 0\ \ 1\ \ 0]^T$, then based on Definitions 4.11, 4.12, and 4.13, the set of two possible combinations of the column 2 of the arbitration matrix is $a_{n=1:5,m=2}(1) = [0\ \ 0\ \ 0\ \ 1\ \ 0]^T$ and $a_{n=1:5,m=2}(2) = [1\ \ 0\ \ 0\ \ 0\ \ 0]^T$. In other words, in each time stage $t$, where the arbiter rotates the selection among existing requests, the arbiter can only select one flit from input port. This means, the sum of the column elements in $A$ must be either 0 or 1, or $0 \leq \sum_{n=1}^{N_{inp}} a_{n,m}(t) \leq 1$. Equation (8) (right side) shows an example of the $A(t)$ for the Snapshot 2 in Fig. 3.*

**Definition 5.3.** *A* Tagged Matrix *$R^*(t)$: $r^*_{n,m}(t)$ is a support matrix that is useful to determine whether a flit must be "held" or can be "released" from the input port, and to compute the next routing request matrix $R(t+1)$. For each time stage unit $t$, the* *matrix request $R(t)$ will be updated as presented in (10). The function $\phi$ contains two subfunctions, i.e., $f_{FR*}$: $R(t), A(t) \rightarrow R^*(t)$ contains operator to form tagged matrix $R^*(t)$ and $f_{UPR}$: $R^*(t) \rightarrow R(t+1)$ to update each element in $R(t+1)$.*

$$\begin{aligned} R(t+1) = \phi(R(t), R^*(t), A(t))& \\ f_{FR*}\colon R(t), A(t) \rightarrow R^*(t)& \\ f_{UPR}\colon R^*(t) \rightarrow R(t+1).& \end{aligned} \tag{10}$$

*According to (11) and (12), the form of tagged matrix $R^*(t)$ depends on the current form of the $R(t)$ and $A(t)$.*

$$\textbf{if } n = const. \textbf{ and } \exists m : r_{n,m}(t) \neq a_{n,m}(t) \textbf{ then}$$
$$\forall m\colon r^-_{n,m}(t) = \begin{cases} 1^- & : r_{n,m}(t) = a_{n,m}(t), \\ 1^* & : r_{n,m}(t) \neq a_{n,m}(t), \\ 0 & : r_{n,m}(t) = 0, \end{cases} \tag{11}$$

$$\textbf{if } n = const. \textbf{ and } \forall m : r_{n,m}(t) = a_{n,m}(t) \textbf{ then}$$
$$\forall m\colon r^+_{n,m}(t) = \begin{cases} 0 & : r_{n,m}(t) = 0, \\ 1^+ & : r_{n,m}(t) = a_{n,m}(t). \end{cases} \tag{12}$$

If we compare matrices in (8), then according to (11), we can see that there are two elements which do not match each other, i.e., $r_{1,2}(1)$ does not match with $a_{1,2}(1)$, and $r_{4,3}(1)$ does not match with $a_{4,3}(1)$. Therefore, these two elements are tagged with the symbol ($^*$) as presented in (14). If minimal one element of row $n$ is tagged with ($^*$), then the other elements having the value 1 in same row $n$ will be marked with the symbol ($^-$). As presented in (14), the elements $r^*_{1,3}(1)$, $r^*_{4,2}(1)$, and $r^*_{4,5}(1)$ are assigned with ($^-$). The other elements of the row $n$ having no 1-element, being tagged with symbol ($^*$) or ($^-$), are assigned with symbol ($^+$) in accordance with (12). As presented in (14), all elements in row 3, i.e., $r^*_{3,1}(1)$ and $r^*_{3,4}(1)$ are assigned with ($^+$), because there is no element in the row 3 having tag symbol ($^*$).

$$R^*_{N_{inp}\times N_{outp}}(t) = \begin{pmatrix} \forall m\colon r^-_{1,m}(t) & or & r^+_{1,m}(t) \\ \forall m\colon r^-_{2,m}(t) & or & r^+_{2,m}(t) \\ \cdots \quad \cdots & & \cdots \\ \forall m\colon r^-_{n,m}(t) & or & r^+_{n,m}(t) \end{pmatrix}, \tag{13}$$

$$R^*(1) = \begin{pmatrix} r^-_{1,m}(1) \\ r^+_{2,m}(1) \\ r^+_{3,m}(1) \\ r^-_{4,m}(1) \\ r^+_{5,m}(1) \end{pmatrix} = \begin{pmatrix} 0 & 1^* & 1^- & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1^+ & 0 & 0 & 1^+ & 0 \\ 0 & 1^- & 1^* & 0 & 1^- \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{14}$$

**Definition 5.4.** *The* Hold/Release Tagging Policy *can be applied by observing whether the row array element of (13) falls in the case according to (11), i.e., $r^-_{n,1:N_{outp}}(t)$, or in the case according to (12), i.e., $r^+_{n,1:N_{outp}}(t)$. Both (11) and (12) comprise an* antecedence *or* condition *part and a* consequence *part.*

**Definition 5.5 (Data Hold Policy).** *If the array elements of the row $n$ of the Tagged Matrix $R^*(t)$ are $r^-_{n,1:N_{outp}}(t)$, then the flit coming from an input port $n$ ($n = const,$) must be held in the input port $n$, because at time stage $t$ and $\forall m, n = const. \Rightarrow \exists m\colon r_{n,m}(t) \neq a_{n,m}(t)$ or there is minimal one element of the $r_{n,1:N_{outp}}(t)$ that has not been granted to be switched out to the requested output port. According to (11), the nongranted element is tagged with symbol ($^*$), while the granted element is tagged with symbol ($^-$). Therefore, according to (15), the next request at time stage $t+1$ of granted element will be dropped to avoid improper multicast flit replication.*

$$\textbf{if } n = const. \textbf{ and } r^*_{n,m}(t) = r^-_{n,m}(t) \textbf{ then}$$
$$\forall m\colon r_{n,m}(t+1) = \begin{cases} 0 & : r^*_{n,m}(t) = 0 \; or \; 1^-, \\ 1 & : r^*_{n,m}(t) = 1^*. \end{cases} \tag{15}$$

**Definition 5.6 (Data Release Policy).** *If the array elements of the row $n$ of the Tagged Matrix $R^*(t)$ are $r^+_{n,1:N_{outp}}(t)$ or all elements of the $r_{n,1:N_{outp}}(t)$ has been granted to be switched out to the requested output ports, then the flit coming from the input port $l = n$ can be released from the input port $n$. In (16), we can define that if the flit from input port $n$ is released from the input port and switched to the output port, then the next considered flit at time stage $t+1$ maybe 1) a zero flit (not a data flit, $r_{n,1:N_{outp}}(t+1) = \emptyset$), 2) a flit of different message with different unicast or multicast output routing direction ($r_{n,1:N_{outp}}(t+1) \neq r_{n,1:N_{outp}}(t)$), or 3) a flit that belongs to the flit that has been released from the input port ($r_{n,1:N_{outp}}(t+1) = r_{n,1:N_{outp}}(t)$).*

$$\begin{aligned} &\textbf{if } n = const. \textbf{ and } r^*_{n,m}(t) = r^+_{n,m}(t) \textbf{ then} \\ &\qquad \forall m\colon r_{n,m}(t+1) = r^{F_{next}}_{n,m}(t+1), \end{aligned} \tag{16}$$

$$R(2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; A(2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{17}$$

$$R^*(2) = \begin{pmatrix} r^+_{1,m}(2) \\ r^+_{2,m}(2) \\ r^+_{3,m}(2) \\ r^+_{4,m}(2) \\ r^+_{5,m}(2) \end{pmatrix} = \begin{pmatrix} 0 & 1^+ & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1^+ & 0 & 0 & 1^+ & 0 \\ 0 & 0 & 1^+ & 0 & 1^+ \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{18}$$

**Lemma 3.** *By using the "hold/release tagging policy" defined in Definition 5.4, improper multicast flit replication on every router can be avoided.*

**Proof of Lemma 3.** If the number of requests of a flit coming from input port $n$ is defined as $N^{req}_{s,n}$ such that at time stage $t = t_s$, $N^{req}_{s,n} = \sum_{m=1}^{N_{outp}} r_{n,m}(t_s)$, then according to (15), a routing request $r_{n,m}(t)$ that has been granted will be reset at the next time stage $t = t_s + 1$. Therefore, every routing request $r_{n,m}(t)$ will be only forwarded once to the output port. □

## 5.3 Proof of the New Theory

**Postulate 1.** *If unicast packets are routed in a certain router such that $\forall n$: $\sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1$ (See Definition 5.1), or if multicast packets are routed in a certain router such that $\exists n$: $\sum_{m=1}^{N_{outp}} r_{n,m}(t) > 1$, and there is no contention between them to access output link such that $\forall m$: $\sum_{n=1}^{N_{inp}} r_{n,m}(t) \leq 1$, then according to Definitions 5.2, 4.11, and 4.12, $\because \forall m$: $T_{s,m} = 1 \Rightarrow \forall t$: $A(t+1) = A(t)$. Thus, according to Definition 5.3, then $\nexists r^*_{n,m}(t) = 1^* \therefore \forall t$: $R(t) = A(t)$ or there is no*

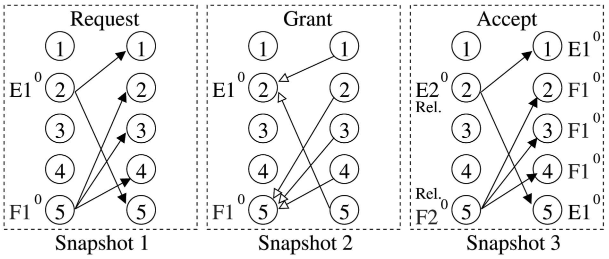

Fig. 7. Scheduling multicast requests without contention.

*need to apply for the "Hold/Release Rule." All unicast packets (without contention) can be released from every input port, or will not be withheld at every input port at each time stage t. Fig. 7 shows an example of the multicast requests in a router without contention.*

**Postulate 2.** *If unicast packets are routed in a certain router such that $\forall n$: $\sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1$ (See Definition 5.1), and there are one or more contentions between them to access output link such that $\exists m$: $\sum_{n=1}^{N_{inp}} r_{n,m}(t) > 1$, according to Definitions 5.2, 4.11, and 4.12, the contention on each output port $m$ can be solved at $t = T_{s,m}$ where $\forall m$: $\exists T_{s,m}$: $r_{n,m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{n,m}(t)$, where $\forall m$: $1 \leq T_{s,m} \leq N_{inp}$. Furthermore according to Definition 5.3, $\because \forall n$: $\sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1 \Rightarrow \forall t,n,m$: $\nexists r^*_{n,m}(t) = 1^-$, or $\forall t,m,n$: $r^*_{n,m} = r^+_{n,m}$ according to (13). Therefore, the "Hold/Release Tagging Policy" is only partially applied, i.e., a unicast request $r_{n,m}(t)$ that has not been granted at time stage t must wait in the input port $l = n$. Fig. 8 shows an example of this situation and how the unicast contention is solved.*

Now, we can extend the problem in a situation where multicast packets are routed in a router with multicast requests such that $\exists n$: $\sum_{m=1}^{N_{outp}} r_{n,m}(t) > 1$, and there are one or more contentions between the multicast requests to acquire the same output ports or $\exists m$ : $\sum_{n=1}^{N_{inp}} r_{n,m}(t) > 1$. This extended problem is actually the key problem that will be solved by our new theory to perform deadlock-free multicast routing without implementing virtual channels in a wormhole-switched NoC. Therefore, instead of introducing a new lemma and its proof, we will introduce new theorems and their proofs as described in the following.

**Theorem 1.** *The Id-field being part of every flit allows to implement a flit-by-flit arbitration and an Id-based routing for interleaving different packets in the same queue, where flits belonging to the same packet have the same Id-tag on every local communication link. Hence, multicast deadlock problem can be solved at each router by further applying a "Hold/Release Tagging Policy" to control and manage conflicting multicast requests.*

**Theorem 2.** *If the multicast dependency and deadlock problems can be solved at each router as mentioned in Theorem 1, then multicast deadlock configurations in the network can be solved, if: 1) the routing algorithm used to route the unicast and multicast packets does not perform cyclic dependency, and 2) a data dropping mechanism at each outgoing communication link is applied to packets that cannot be assigned to an Id slot on the communication link, or the mechanism can be neglected if sufficient number of Id slots is set per link for considered traffics.*

**Proof of Theorem 2.** The Theorem 2 can be proved, if Theorem 1 can be proved and the conditions mentioned in Theorem 2 are fulfilled. Therefore, we will explain and prove the need for the necessary conditions mentioned in the Theorem 2 as follows:

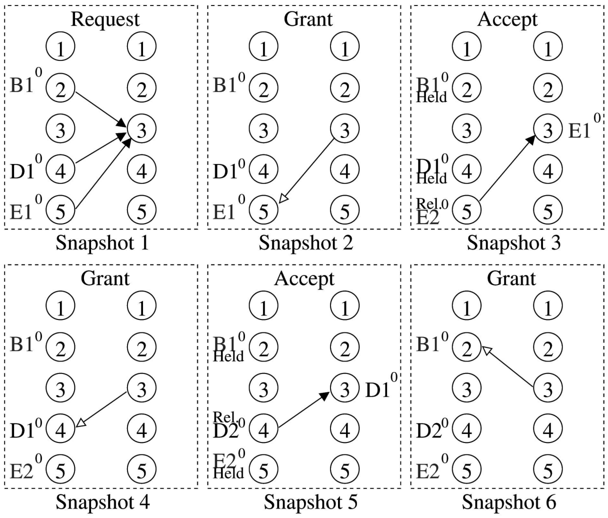

Fig. 8. Scheduling unicast requests with contention.

1. The necessary condition mentioned by the item 1) in the Theorem 2 is needed because routing algorithm used to route unicast and multicast packets are the same according to the proposed hardware solution in this paper. Therefore, if the used routing algorithm does not perform cyclic dependency, then the proposed tree-based multicast routing is also free from deadlock configuration. The proof of the deadlock-freeness in term of the cyclic dependency problem is presented in detail in [21], [22], and [23].
2. The necessary condition mentioned by the item 2) in the Theorem 2 is required because if the data flits are not dropped then they will stall in the router especially if they must wait other messages to free one Id slot for very long time. In this case, the data flits will be stagnant and occupy many buffers in the upstream channels, and do not give spaces for other messages to flow (chain block). However, if the number of available Id slots per link is set, for example, equal to (5) (when using minimal fully adaptive routing), then the packet stall can be avoided because each considered traffic can reserve one Id slot to flow on every communication link.

□

**Proof of Theorem 1.** The circulating arbitration mechanism can guarantee that one flit of unicast or multicast packets can be forwarded to each outgoing link at each router node, where multicast conflict may occur. After arbitration process at each time $t$, a hold-release tagging mechanism can also guarantee that improper replication of the multicast packets can be avoided, because: 1) the granted multicast bit-requests will be assigned and will not be included again in the next arbitration process, and 2) the flits having multicast bit-requests will be kept in the FIFO queue until all its multiple bit-requests are granted.

The circulating selection result of the arbitration process at each output port maybe random and not uniform. Therefore, there are two possible configurations after the arbitration process, i.e., 1) all requests of a multicast flit from an input port $n$ are granted at the same time stage $t$ ($r_{n,1:N_{outp}}(t) = a_{n,1:N_{outp}}(t)$), or 2) not all the multicast requests from an input port $n$ are granted ($\forall h \in \varphi_n^{req}$: $\exists r_{n,h}(t) \neq a_{n,h}(t)$). In the situation 1), the multicast flit can be released from FIFO queue, and in the situation 2), the multicast flit must be held in FIFO queue, and the hold-release tagging policy and the circulating/rotating flit-by-flit arbitration will then cover the situation.

By circulating the bit-set selection in every column $m$ of $A(t)$ at each time stage $t$, where the circulating combinations of $a_{1:N_{inp},m}(t)$ for $\forall m$ (all output ports) are independent each other, then it is possible, in finite time $T_f$ to find $A(T_f)$ in such a way, that all conflicting multicast flits can escape from multicast dependency.

1. If the amount of requests in every output port $m$ at $t = t_s$ is $N_{s,m}^{req}$ such that $1 \leq N_{s,m}^{req} \leq N_{inp}$ (See (3)), then the required number of circulating arbitration time to grant the request from the input port $l \in \Phi_m^{req}$ to output $m$ appear at $t_s$ is $T_{s,m} = N_{s,m}^{req}$. The probability that the request $r_{l,m}(t_s)$ is selected by grant acknowledge $a_{l,m}(t_s)$ is $Prob(r_{l,m}(t_s) = a_{l,m}(t_s)) = \frac{1}{N_{s,m}^{req}}$. According to Definitions 4.10, 4.11, 4.12, and 4.13, then at $t = T_{s,m}$ we will achieve that $r_{1:N_{inp},m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$. The maximum number of requests to an output port $m$ is $N_{inp}$. Hence, if $\Phi_m^{req} \subseteq \Phi$, then $r_{1:N_{inp},m}(t_s) = \bigcup_{t=1}^{N_{inp}} a_{1:N_{inp},m}(t)|t_s = 1$.
2. If at $t = t_s$, there is multicast requests from an input port $n$ such that $N_{s,n}^{req} > 1$ (See Definition 4.9), then we obtain a set of input ports $\varphi_n^{req}$ in such a way that $r_{n,h}(t_s) = 1$ *iff* an output port $h \in \varphi_n^{req}$ (See (2)). The probability that every single request of the multicast request $r_{n,h}(t_s)$ from the input port $n$ is selected by the arbitration unit at the requested output port $h \in \varphi_n^{req}$, depends also on the number of requests from other input ports in the set $\Phi_h^{req}$ to the same output port $h$.

In accordance with items 1 and 2 mentioned above, then we can derive a conditional equation such that the multicast deadlock problem is solved as described in the following equation:

$$R(t_s) = \left( \bigcup_{t=t_s}^{T_{s,1}} a_{1:N_{inp},1}(t) \cdots \bigcup_{t=t_s}^{T_{s,N_{outp}}} a_{1:N_{inp},N_{outp}}(t) \right). \qquad (19)$$

The typical contentionless switching situations presented in Fig. 7 can be solved at every single time stage such that $\forall t, m, n : r_{n,m}(t) = a_{n,m}(t)$. The typical problem presented in Fig. 8 can be solved per output port basis, where at every output port the problem is solved at $t = T_{s,m}$ such that $\forall m : r_{1:N_{inp},m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$. The multicast contention problem presented, e.g., in Fig. 3 must be solved per input-output basis because of the existing multicast requests.

Because the circulating arbitration order at every input port $m$ is not uniform or independent from each other, then there will be many possible combinations of $A(t)$ at every time stage $t$. The arbitration time solution $T_{s,m}$ at every output $m \in \varphi$ may vary and depends on the number of requests $N_m^{req}$ to the output port $m$ ($T_{s,m} = N_m^{req}$). However, we can guarantee that, at the maximum time of $t = T_f$ such that $\forall m$: $T_f = T_{s,m}^{max} = max(T_{s,m})$ or $T_f = max(T_{s,1}, T_{s,2}, \ldots, T_{s,N_{outp}})$, then the multicast deadlock dependency problem on each router is solved at $T_f if$ there is no congestion in the outgoing links, $\because at\ t = T_f \Rightarrow \forall m : \bigcup_{t=t_s}^{T_f} a_{1:N_{inp},m}(t) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$.

Therefore, the conditional equation (19) is fulfilled, and by following the Proofs of Lemma 1, 2, and 3, then the multicast deadlock and dependency problems on each router is solved without improper multicast flit replication. In other words, all requests depicted in $R(t_s)$ (at initial time $t_s$) will finally receive a grant acknowledge $A(T_f)$ in such a way that all flits appear at $t = t_s$ from all input ports $n \in \Phi$ would have been switched out to the output ports $m \in \varphi$ or would have been rescued from the multicast contention in the router in finite time stage $T_f$ where $1 \leq T_f \leq N_{inp}$. If congestion occurs at any outgoing link then the solution is postponed for $T_{wf}$ time stage. Hence, the problem is solved at $t = T_f + T_{wf}$, where $T_{wf}$ is the number of time stages to wait for a free data slot available in the queue of the congested link connected directly to the most requested output port.

The descriptions given above have proved the Theorem 1 because the multicast problem can be solved in such a way that the contenting or conflicting multicast packet can be rescued from the multicast dependency in a router. If the multicast dependency (deadlock) problem can be solved on every router $R_c \in \Re$, then the network is free from multicast deadlock problem as long as the routing algorithm used to route unicast and multicast packets does not form cyclic dependencies. The detailed proof of the last statement can be found in [21], [22], [23].

$T_f$ depends on the concrete multicast conflict situation in each router. For instance, in the multicast conflict case presented in Fig. 3, the flit coming from the $PORT\ 3$ port can be rescued from the multicast-dependency after generating one in-column bit-set combination of the arbitration matrix $A(1)$ as shown in Snapshot 2. While the flits coming from $PORT\ 1$ and $PORT\ 4$ ports can be rescued after generating two in-column bit-set combinations of the arbitration matrices $A(1)$, and $A(2)$ as shown in Snapshots 2 and 4, respectively. From Fig. 3, we can see that for $N_{inp} = N_{outp} = 5$, then the numbers of request at every output port $m \in \{1, 2, 3, 4, 5\}$ are $N_{s,1}^{req} = N_{s,4}^{req} = N_{s,5}^{req} = 1$ and $N_{s,2}^{req} = N_{s,3}^{req} = 2$. Thus, $\forall m = \{1, 2, 3, 4, 5\}$ then

$$T_f = max(T_{s,1}, T_{s,2}, T_{s,3}, T_{s,4}, T_{s,5}) = max(1, 2, 2, 1, 1) = 2.$$

Therefore, the multicast dependency deadlock depicted in Fig. 3 can be solved in the next finite time stage $T_f = 2$. The rotating output selection per output port in four successive time stages of the problem shown in Fig. 3 is presented in the following tabular.

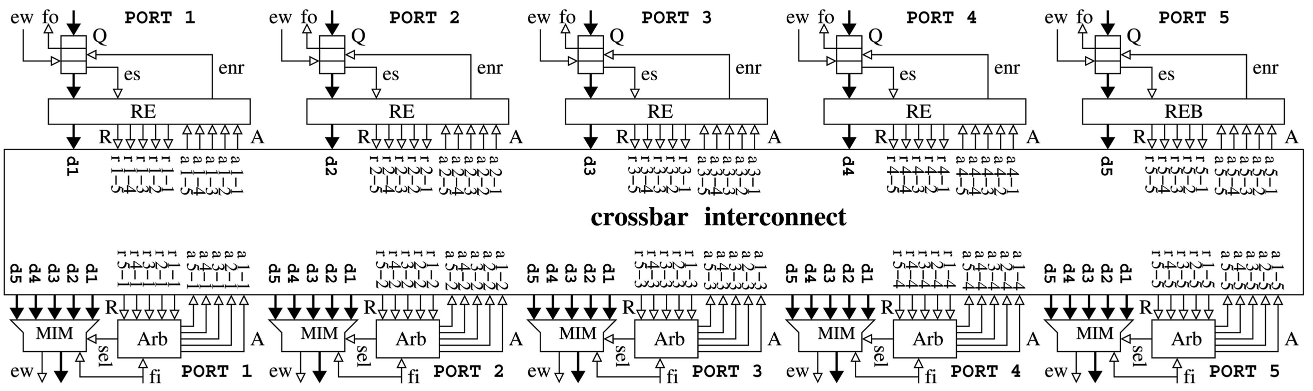

Fig. 9. An example of typical five-port router/switch matrix structure based on the proposed hardware solution.

| $t$ | $m=1$ | $m=2$ | $m=3$ | $m=4$ | $m=5$ |
|---|---|---|---|---|---|
| 1 : | $a_{3,1}(1)$ | $a_{4,2}(1)$ | $a_{1,3}(1)$ | $a_{3,4}(1)$ | $a_{4,5}(1)$ |
| 2 : | $a_{3,1}(2)$ | $a_{1,2}(2)$ | $a_{4,3}(2)$ | $a_{3,4}(2)$ | $a_{4,5}(2)$ |
| 3 : | $a_{3,1}(3)$ | $a_{4,2}(3)$ | $a_{1,3}(3)$ | $a_{3,4}(3)$ | $a_{4,5}(3)$ |
| 4 : | $a_{3,1}(4)$ | $a_{1,2}(4)$ | $a_{4,3}(4)$ | $a_{3,4}(4)$ | $a_{4,5}(4)$ |
| ... | ... | ... | ... | ... | ... |
| | $T_{s,1}=1$ | $T_{s,2}=2$ | $T_{s,3}=2$ | $T_{s,4}=1$ | $T_{s,5}=1$ |

□

## 5.4 Switch Architecture

The most interesting aspect of the new theory, is that the VLSI architecture which implement the new theory can be designed in a regular modular switch architecture. Fig. 9 shows us a typical five-port generic VLSI architecture of a switch that represents the methodology. The upper part of Fig. 9 presents modular units at the input port of the switch router, while the lower part presents modular units at the output port. In the figure, we can see that $\forall n, m = 1, 2, 3, 4, 5$: $\exists r_{n-m}, a_{n-m}$. The control paths of signals $r_{n-m}$ and $a_{n-m}$ are practically similar to the variables $r_{n,m}(t)$ and $a_{n,m}(t)$ defined theoretically in Definitions 5.1 and 5.2, respectively. The logical function to form the Tagged Matrix defined in Definition 5.3 are practically implemented in the $RE$ unit at every input port. The Arbiter ($A$) units at every input port represent the Arbitration Matrix defined in Definition 5.2 and perform the rotating flit-by-flit arbitration described in Algorithm 2. The Routing Table $T(k)$ defined in Definition 4.7 and the ID Slot Table $S(k)$ defined in Definition 4.14 are implemented in the $RE$ unit at every input port and in the $MIM$ unit at every output port, respectively.

Fig. 9 is only a representative figure of many possible architectures that has four-stage pipeline, i.e., input buffering, routing, arbitration, and link traversal stages. Pipeline stages number can be freely determined by NoC designers. The input buffering is implemented using a standard first-in first-out algorithm. The main critic of the NoC router design is the implementation of two tables on every one-directional link that can lead to area overhead. However, when a number of $N$ messages is allowed in-flight in the same physical link, then compared to a VC-based solution, our concept requires theoretically less logic area and power, since the size of the two tables having $N$ slot registers will be less than the size of $N$ number of VC buffers.

This paper focuses on the proof of the new theory for deadlock-free multicast routing in a wormhole-switched virtual channelless NoC. The verification through RTL-simulation as well as the early VLSI architecture implementations using CMOS standard-cell technology can be found in [24] in 2D $4 \times 4$ Mesh NoC with static XY routing algorithm, in [25] in 2D $8 \times 10$ Mesh NoC with static and partial adaptive routing algorithms, and in [26] in 2D $8 \times 10$ Mesh-Planar NoC with 2D planar adaptive routing algorithm.

## 6 Conclusions

This paper has presented a new theory and methodology for a deadlock-free tree-based multicast routing suitable for NoCs. The multicast routing theory and methodology is supported by allowing flits of different wormhole messages being interleaved/multiplexed at flit-level on the same link. Multicast deadlock problems are solved and improper multicast replications are prevented by using the hold-release tagging mechanism. The message routing is guaranteed correct because flits belonging to the same wormhole message are allocated to the same Id slot on a certain link. The introduction of the local Id slots can be interpreted implicitly as a mechanism to reduce the size of the RRT. The size of ID-tags in the RRT can be set less than the number of node entries in the NoC to run specific data distribution scenarios. This case has been experimented in [25] and [26] by setting 16 ID-tags per link in the 80 cores (mesh $8 \times 10$) NoC.

Fortunately, the traffics in embedded MPSoCs are predictable, and it is a rare case that more than 15 messages are in-flight in the same link. However, the packet dropping mechanism must be applied in this case to avoid data-flow stall. If the number of ID-tags per link $N_{slot}$ is set to cover all considered traffics, e.g., equal to (5), then the packet dropping mechanism can be neglected. There is a design trade-off in this aspect. By setting the minimum $N_{slot}$ per link as discussed in Section 4.6, the size of the RRT and ID Slot Table units would be larger, but there is no need for a retransmission protocol. When data dropping is applied and the number of entries in the tables units is reduced, then router size will be smaller, but the retransmission protocol must be applied leading to area overhead in the network interface, and probably time overhead when the data drop occurs.

## APPENDIX (NOTATIONS)

| Symbol | | Description |
|---|---|---|
| $N_{node}$ | : | Number of router node in a NoC |
| $\Re$ | : | Set of routers $\{R_1, R_2, \ldots, R_{N_{node}}\}$ |
| $R_c$ | : | Router node $c$, where $R_c \in \Re$ |
| $\Lambda$ | : | Set of communication links in a NoC |
| $L_{i,j}$ | : | Link connecting $R_i$ to $R_j$, where $L_{i,j} \in \Lambda$ |
| $N_{inp}$ | : | Number of input ports in a router |
| $N_{outp}$ | : | Number of output ports in a router |
| $\Phi$ | : | Set with elements $\{1, 2, \ldots, N_{inp}\}$ |
| $\varphi$ | : | Set with elements $\{1, 2, \ldots, N_{outp}\}$ |
| $n$ | : | Input port number, $n \in \Phi$ |
| $m$ | : | Output port number, $m \in \varphi$ |
| $N^{req}_{s,m}$ | : | Number of request to acquire output port $m$ |
| $N^{req}_{s,n}$ | : | Number of request from input port $n$ |
| $N_{slot}$ | : | Number of Id Slot on every link |
| $\Gamma$ | : | Set of elements $\{0, 1, 2, \ldots, N_{slot} - 1\}$ |
| $\varepsilon_{type}$ | : | Set of element $\{header, databody, tail, response\}$ |
| $type$ | : | Type of a flit, $type \in \varepsilon_{type}$ |
| $\Omega$ | : | Set of elements $\{0, 1, 2, \ldots, N_{slot} - 1\}$ |
| $Id, k$ | : | Local Id-tag and Id slot, $k \in \Omega$, $\Omega \subseteqq \Gamma$ |
| $R(t)$ | : | Routing Request Matrix |
| $r_{n,m}(t)$ | : | Matrix element of $R(t)$ |
| $A(t)$ | : | Arbitration (Routing Acknowledge) Matrix |
| $a_{n,m}(t)$ | : | Matrix element of $A(t)$ |
| $R^*(t)$ | : | Tagged Request Matrix |
| $r^*_{n,m}(t)$ | : | Matrix element of $R^*(t)$ |
| $S^k$ | : | State of Id Slot $k$, where $k \in \Omega$ |
| $S(k)$ | : | ID Slot Table |
| $T(k)$ | : | Routing Table |

## ACKNOWLEDGMENTS

The authors gratefully acknowledge the insightful comments and suggestions made by the reviewers. Special thanks are given to Deutcher Akademischer Austausch-Dienst (DAAD, German Academic Exchange Service) awarding the DAAD-Scholarship for Faizal A. Samman to pursue the engineering doctoral degree at Darmstadt University of Technology in Germany. The authors would also like to thank LOEWE-Zentrum AdRIA in Fraunhofer Institute LBF Darmstadt for further cooperation and for possible implementation of the concept and theory presented in this paper within Project AdRIA (Adaptronik-Research, Innovation, Application) funded by Hessian Ministry of Science and Arts with grant number IIIL4-518/14.004 (2008).

## REFERENCES

[1] G. Fox et al., "Fortran D Language Specification," Technical Report CRPC-TR 90079, Center for Research on Parallel Computation, Rice Univ., Dec. 1990.

[2] J. Merlin, "Techniques for the Automatic Parallelization of Distributed Fortran 90," Technical Report SNARC 92-02, Southampton Univ., Nov. 1991.

[3] High Performance Fortran Forum, "High Performance Fortran Language Specification, Version 1.0," *Scientific Programming,* vol. 2, no. 1, May/June 1993.

[4] Message Passing Interface Forum, "MPI-2: Extensions to the Message-Passing Interface," technical report, Univ. of Tennessee, Nov. 2003.

[5] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos, "PVM and MPI: A Comparison of Features," *Calculateurs Paralleles,* vol. 8, no. 2, pp. 137-150, May 1996.

[6] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing,* MIT Press, http://www.csm.ornl.gov/pvm, 1994.

[7] X. Lin, P.K. McKinley, and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 8, pp. 793-804, Aug. 1994.

[8] J. Duato, "A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 9, pp. 976-987, Sept. 1995.

[9] R.V. Boppana, S. Chalasani, and C.S. Raghavendra, "Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 6, pp. 535-549, June 1998.

[10] M. Barnett, D.G. Payne, R.A. van de Geijn, and J. Watts, "Broadcasting on Meshes with Worm-Hole Routing," *J. Parallel and Distributed Computing,* vol. 35, no. 2, pp. 111-122, 1996.

[11] M.P. Malumbres, J. Duato, and J. Torrelas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors," *Proc. Eighth IEEE Symp. Parallel and Distributed Processing,* pp. 186-189, 1996.

[12] D.R. Kumar and W.A. Najjar, P.K. Srimani, "A New Adaptive Hardware Tree-Based Multicast Routing in K-Ary N-Cubes," *IEEE Trans. Computers,* vol. 50, no. 7, pp. 647-659, July 2001.

[13] P. Abad, V. Puente, and J.-A. Gregorio, "MRR: Enabling Fully Adaptive Multicast Routing for CMP Interconnection Networks" *Proc. 15th IEEE Int'l Symp. High Performance Computer Architecture (HPCA '09),* pp. 355-366, Feb. 2009.

[14] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient Unicast and Multicast Support for CMPs," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture (MICRO '08),* pp. 364-375, Nov. 2008.

[15] L. Wang, Y. Jin, H. Kim, and E.J. Kim, "Recursive Partitioning Multicast: A Bandwidth-Efficient Routing for Networks-on-Chip," *Proc. Third ACM/IEEE Int'l Symp. Networks-on-Chip (NOCS '09),* pp. 64-73, May 2009.

[16] N.E. Jerger, L.-S. Peh, and M. Lipasti, "Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support," *Proc. 35th Int'l Symp. Computer Architecture (ISCA '08),* pp. 229-240, June 2008.

[17] A.S. Vaidya, A. Sivasubramaniam, and C.R. Das, "Impact of Virtual Channels and Adaptive Routing on Application Performance," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 2, pp. 223-237, Feb. 2001.

[18] K. Aoyama and A.A. Chien, "The Cost of Adaptivity and Virtual Lanes in a Wormhole Router," *J. VLSI Design,* vol. 2, no. 4, pp. 315-333, 1995.

[19] D.A. Ilitzky, J.D. Hoffman, A. Chun, and B.P. Esparza, "Architecture of the Scalable Communications Core's Network on Chip," *IEEE Micro,* vol. 27, no. 5, pp. 62-74, Sep./Oct. 2007.

[20] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S.W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro,* vol. 27, no. 5, pp. 41-50, Sep./Oct. 2007.

[21] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *Proc. 19th Int'l Symp. Computer Architecture,* pp. 278-287, 1992.

[22] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers,* vol. C-36, no. 5, pp. 547-553, May 1987.

[23] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 12, pp. 1320-1331, Dec. 1993.

[24] F.A. Samman, T. Hollstein, and M. Glesner, "Multicast Parallel Pipeline Router Architecture for Network-on-Chip," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE '08),* pp. 1396-1401, 2008.

[25] F.A. Samman, T. Hollstein, and M. Glesner, "Adaptive and Deadlock-Free Tree-Based Multicast Routing for Networks-on-Chip," *IEEE Trans. Very Large Scale Integration Systems,* vol. 18, no. 7, pp. 1067-1080, July 2010.

[26] F.A. Samman, T. Hollstein, and M. Glesner, "Planar Adaptive Router Microarchitecture for Tree-Based Multicast Network-on-Chip," *Proc. First Int'l Workshop Network-on-Chip Architecture (NoCArch '08), in conj. with the IEEE/ACM Int'l Symp. Microarchitecture (MICRO-41),* pp. 6-13, 2008.

**Faizal Arya Samman** received the bachelor of engineering degree (with Honour) in electrical engineering from Gadjah Mada University at Yogyakarta, Indonesia in 1999 and the master of engineering degree from Inter University Center for Microelectronics Research, Bandung Institute of Technology in Indonesia in 2002 with Scholarship Award from Indonesian Ministry of National Education in Control and Computer System Laboratory. In 2002, he was appointed to be a research and teaching staff at Hasanuddin University in Makassar, Indonesia. From 2006 until 2010, he received scholarship award from Deutscher Akademischer Austausch-Dienst (DAAD, German Academic Exchange Service) to pursue the engineering doctoral degree at Darmstadt University of Technology, in Germany. He is now working toward the postdoctoral program in LOEWE-Zentrum AdRIA (Adaptronik-Research, Innovation, Application) within the research cooperation framework between Darmstadt University of Technology and Fraunhofer Institute LBF in Darmstadt. His research interests include network on-chip (NoC) microarchitecture, NoC-based multiprocessor system-on-chip application mapping, programming models for multiprocessor systems, design and implementation of analog and digital electronic circuits for control system applications on FPGA/ASIC as well as energy harvesting systems and wireless sensor networks. He is a member of the IEEE.

**Thomas Hollstein** received the graduation in electrical engineering/computer engineering from Darmstadt University of Technology in 1991 and the PhD degree on "Design and interactive Hardware/Software Partitioning of Complex Heterogeneous Systems" from Darmstadt University of Technology in 2000. In 1992, he joined the research group of the Microelectronic Systems Lab at Darmstadt University of Technology. He worked in several research projects in neural and fuzzy computing and industrial VHDL-based design. Since 1995, he focused his research on hardware/software codesign. Since 2000, he is working as a senior researcher, leading a research group focusing System-on-Chip communication architectures, the design of reconfigurable HW/SW Systems-on-Chip and integrated SoC test and debug methodologies. His current research interests are in the fields of networks-on-chip, hardware/software co-design, systems-on-chip design, printable organic and inorganic electronics, and RFId circuit and system design. Furthermore, he is giving lectures on VLSI design and CAD methods. From 2001 until now, he has been member of a leader team initiating and establishing a new international master program in "Information & Communication Engineering" at Darmstadt University of Technology. In 2010, he was appointed as a professor at Tallin University of Technology in Department of Computer Engineering, Dependable Embedded Systems Group.He is a member of the IEEE.

**Manfred Glesner** received the diploma and PhD degrees from Saarland University, Saarbrücken, Germany, in 1969 and 1975, respectively, and the three Doctor Honoris Causa degrees from Tallinn Technical University, Estonia, in 1996, Polytechnic University of Bucharest, Romania, in 1997, and Mongolian Technical University, Ulan Bator, in 2006. His doctoral research was based on the application of nonlinear optimization techniques in computer-aided design of electronic circuits. Between 1969 and 1971, he has researched work in radar signal development for the Fraunhofer Institute in Werthoven/Bonn, Germany. From 1975 to 1981, he was a lecturer in the areas of electronics and CAD with Saarland University. In 1981, he was appointed as an associate professor in electrical engineering with the Darmstadt University of Technology, Germany, where, in 1989, he was appointed as a full professor for microelectronic system design. His current research interests include advanced design and CAD for micro- and nanoelectronic circuits, reconfigurable computing systems and architectures, organic circuit design, RFID design, mixed-signal circuit design, and process variations robust circuit design. With the EU-based TEMPUS initiative, he built up several microelectronic design centers in Eastern Europe. Between 1990 and 2006, he acted as a speaker of two DFG-funded graduate schools. He is a member of several technical societies and he is active in organizing international conferences. Since 2003, he has been the vice president of the German Information Technology Society (ITS) in VDE and also a member of the DFG decision board for electronic semiconductors, components, and integrated systems. He was a recipient of the honor/decoration of "Palmes Academiques" in the order of Chevalier by the French Minister of National Education (Paris) for distinguished work in the field of education in 2007/2008. He is a fellow of the IEEE.