Received 26 June 2022, accepted 10 August 2022, date of publication 17 August 2022, date of current version 29 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3199671

RESEARCH ARTICLE

Accelerating NoC Verification Using a Complete Model and Active Window

SURAJIT DAS[1], **CHANDAN KARFA**[2], **(Senior Member, IEEE), AND SANTOSH BISWAS**[3], **(Senior Member, IEEE)**

[1]Department of Computer Science and Engineering, GITAM School of Technology, Bengaluru, Karnataka 561203, India
[2]Department of Computer Science and Engineering, IIT Guwahati, Guwahati, Assam 781039, India
[3]Department of Electrical Engineering and Computer Science, IIT Bhilai, Raipur, Chhattisgarh 492015, India

Corresponding author: Surajit Das (surajitdas3020@gmail.com)

ABSTRACT This work presents formal modeling of Network-on-Chip (NoC) considering detailed functional units of NoC. The intricate modeling of NoC router components like buffer, switch, and arbiter is accomplished using Finite State Machine (FSM). As in the case of a real NoC, parallel execution of these functional units is carried out by maintaining the synchronization between these functional units within a router and between each of the adjacent routers in the presented formal model. Important properties for the correctness of the proposed model are verified using a model checker. Implementing a detailed and a complete NoC model is a memory extensive operation while verifying with a model checker. We have introduced a concept of active windows to verify each router and its communication with the adjacent routers. The correctness of the model is checked by verifying the synchronization between NoC functional units and NoC routers, verifying progress in functional units, and verifying the successful transfer of packets. Verification of starvation freedom in an NoC router is also performed for round-robin arbiter and fixed-priority arbiter. Parallel threads are used in the experiments to reduce the verification time.

INDEX TERMS Network-on-chip, formal model, model checking, finite state machine, starvation.

I. INTRODUCTION

Modeling NoC components at the detail level are essential for formal verification to ensure that an NoC works correctly after manufacturing the actual hardware. Modeling NoC close to hardware functionality and scalability of the verification method are two primary challenges in NoC verification. In most of the existing NoC verification works, detailed modeling of the complete NoC system is missing [1], [2], [3], [4], [5], [6], [7], [8]. We have considered NoC functional units in a comprehensive way in this work. It is convenient to implement an NoC model with Finite State Machine (FSM) in a model checker like NuSMV [9]. Therefore, we have chosen FSM for modeling NoC so that the model can be easily encoded and various properties can be checked using a state-of-the-art model checker.

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

A. NoC ROUTER COMPONENTS

A 3×3 mesh NoC is shown in Fig. 1(a). R1, R2, R3, etc., are the routers shown with square boxes. The brown oval-shaped units represent the processors. Each router constitutes of buffers for storing packets, switch for computing the route for a packet and diverting it to the desired output port, and an arbiter to control the transmission of packets via an output port. We have considered NoC functional units or NoC components like buffer, switch and arbiter for modeling NoC. An NoC works with the help of the collaborative efforts between all these units. *Buffers* are present at the input ports of a router. The five bidirectional ports in a router are shown in Fig. 1(b). They are used as temporary storage for a packet before being transmitted in the desired path. The routing direction for a packet is computed by another functional unit called *switch*. After the route computation is over, the packet requests for the expected output port. There may be multiple packets competing for the same output port simultaneously. An *arbiter* is present at each output port and resolves the

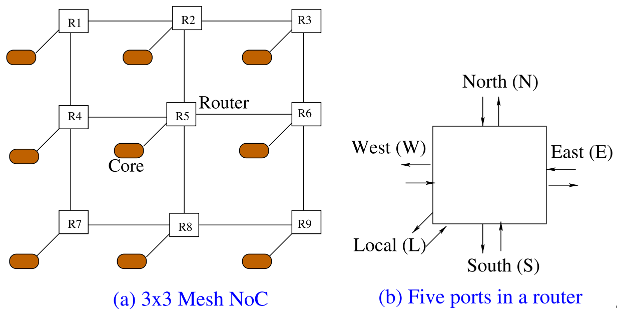

FIGURE 1. A 3 × 3 Mesh NoC and five bidirectional ports in a router.

conflict by selecting one packet from the competing packets. The packet selected by an *arbiter* is transmitted towards the next router.

An output port and its connection to the adjacent router are considered as a resource of conflict in an NoC. When multiple packets from different input ports continuously try to access the same output port in a router, the competing packets from all input ports should get a fair chance to be transmitted eventually. This is called starvation-freedom. Starvation-freedom ensures fairness in selecting packets for transmission from all the input ports.

B. CONTRIBUTIONS

This work is an extension to our prior work [10] where we have presented modeling of an NoC using FSM and verification of starvation considering each router at a time. In this work, we have considered the synchronization between routers as well while implementing our model using NuSMV model checker. Since considering the complete NoC results in state space explosion, we have introduced the concept of active windows in this work. Besides verifying starvation freedom with active windows, we have verified synchronization between router components, progress in individual router components and transfer of packets between routers in this work. We have encoded our FSM based NoC models using NuSMV model checker [9]. Linear Temporal Logic (LTL) [11] is used to represent the specifications for progress, transfer of packet, synchronization, and starvation-freedom for the verification purpose. Whether a given property is satisfied or not, based on that, the model checker reports True or False (with a debug trace). Specifically, the contributions of this work are as follows:

- Detailed modeling of NoC router functional units like buffer, switch, arbiter and the synchronization between these components using FSMs are presented.
- Communication of a router with its neighbouring routers is implemented in NuSMV by introducing the active window concept.
- Demonstrate the designing of fixed-priority arbiter and round-robin arbiter using FSMs to select a packet from more than one competing packet.
- For checking the correctness of the NoC model, verification of synchronization within a router, progress between NoC components and loss-less transfer of packets are performed.
- As an application of our FSM based model, verification of starvation-freedom on fixed-priority arbiter and round-robin arbiter are demonstrated.
- The verification time is reduced significantly by using parallel threads for individual routers.

The rest of this paper is organized as follows. A brief description of FSM and the short form used for describing the NoC model is presented in Section II. Detailed modeling of NoC components is presented in Section III. Verifying the correctness of the FSM model is described in Section IV. Application of the FSM model and experimental results are presented in Section V and Section VI, respectively. We conclude the work in Section VIII.

II. FINITE STATE MACHINE AND THE NAMING CONVENTION

In this section, we briefly describe Finite State Machine (FSM). The numbers of NoC routers and NoC components vary depending upon the size of the NoC. We consider a 3 × 3 Mesh NoC as a reference NoC for the convenience of presenting our FSM based NoC Model. The same procedure is applied while modeling and implementing NoC of bigger sizes.

A. FINITE STATE MACHINE

The Finite State Machine (FSM) is being used by the research community for modeling of transition systems in various applications [12], [13], [14], [15], [16], [17], [18], [19]. Each FSM has a finite number of states, and a transition takes place from one state to another state based on predefined transition rules. Deterministic Finite Automata (DFA) is a particular class of FSM where each transition is a unique path from one particular state to another particular state with respect to a specific input or with respect to the truth value of a specific condition. A formal definition of a DFA is given below [20].

Definition 1: A Deterministic Finite Automata is represented using a quintuple,

$\{Q,\ \Sigma,\ \delta,\ q_o,\ F\}$.

Here, Q represents a finite set of states that constitute the automata,

Σ represents the input symbol that are used for state transition,

δ represents the transition function $\delta : Q \times \Sigma \rightarrow Q$,

q_o represents the initial state, $q_o \in Q$,

F represents the set of final sets, $F \subseteq Q$.

In this work, corresponding to each NoC functional unit, an FSM is modeled. An FSM state indicates the current status of the corresponding NoC functional unit. Each FSM starts from an initial or starting state q_o, indicated with an incoming arrow. The current state of an FMS keeps on changing with respect to the movement of packet. Once the transmission of a packet is over, the FSMs corresponding to that transmission return to their initial states again. Therefore, we consider the

TABLE 1. Short forms for describing FSM transitions.

Functional Units	Short Forms	Functional Units	Short Forms
Arbiter	Ar	Buffer	Bu
Priority	Pr	Return	Re
Switch	Sw	Sync	Sy

initial state as the final state as well, i.e., $F = \{q_0\}$ in our FSM based NoC modeling. This process continues for the transmission of other packets. This is a continuous process, and the states of FSMs keep on changing.

There present a number of FSMs in a complete NoC model. In an NoC, each of the functional units interacts with some other NoC functional units for the proper functioning of the system. In the same context, each of the FSM interacts with other FSMs in the NoC model. All the transitions in an FSM are controlled by the states of other FSMs with whom the FSM interacts, and all transitions happen in a hand checking fashion. Transitions for the FSMs are described as $\delta : Q \times \Sigma \rightarrow Q$. In the context of our FSM based NoC model, Q represents the states in the FSM under consideration and Σ represents the FSM states information for the complete NoC model. Alternately, Σ represents the global states information of the complete NoC that are used for defining state transitions δ in each individual FSM.

B. SHORT FORMS AND THE NAMING CONVENTION

We have used Fig. 1(a) as a reference while describing the FSM model for different NoC components. We consider the South port of router R2 in Fig. 1(a) while demonstrating FSM models for buffer, switch, and arbiter. The five ports present in a router are shown in Fig. 1(b). While describing buffer and switch at the South port (router R2 in Fig. 1(a)), we are considering the movement of packet towards router R2 from router R5. While describing the arbiter at the South port (router R2 in Fig. 1(a)), we are considering the movement of packets from router R2 toward router R5. Every time we mention about a router in describing the FSM model, that router corresponds to Fig. 1(a).

For accommodating space while describing the transitions in an FSM diagram, we use the short form[1] as shown in Table 1. For a buffer we use "Bu", for an arbiter we use "Ar", for a switch we use "Sw", etc. in the FSM diagrams in this work. Besides these short forms in Table 1, we use L, E, W, N and S for the Local port, the East port, the West port, the North port, and the South port, respectively. The Name of an NoC component is associated with the router name as a prefix and the port name as a postfix to the component name. For example, R2BufferS indicates the buffer associated with the S port of router R2, R5ArbiterN indicates the arbiter present at the North port of router R5, etc. These component names are used in short form in the FSM diagrams. For example, R2BufS is used for R2BufferS, R5ArN is used for R5ArbiterN, and so on as per the Table 1. Thus, the information about the associated router and associated port of NoC components is self-explanatory by using the naming convention for the NoC components.

[1]Short form in Table. 1 are used only in FSM diagrams for representing state transitions for clarity of the figures. We use full forms in other places for better readability.

III. FORMAL MODELING OF NoC USING FSM

In this section, we describe the FSM models of various components of the NoC and their synchronization. We have modeled buffer, switch and arbiter for an NoC router. Synchronization between them is maintained with help of the handshaking principle between a group of FSMs. Considering components from all NoC routers and maintaining the synchronization between them give a complete model for the NoC.

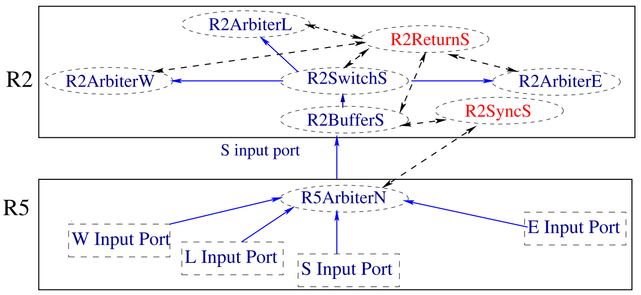

FIGURE 2. Packets moving from router R5 to R2 and synchronization is maintained by return and Sync.

A. HIGH-LEVEL OVERVIEW OF THE MOVEMENT OF PACKETS

We have shown the movement of packets from router R5 to router R2 (of Fig. 1) in the Fig. 2. The possible paths for the movement of packets are shown with solid blue arrows. Movements of packets are shown from the North output port of router R5 to the South input port of router R2. An arbiter named R5ArbiterN is present at the North output port of router R5, as shown in the Fig. 2. The packets from four input ports, namely the Local, East, West and South, can transmit packets via the North output port of router R5. If packets from more than one input port have to compete for a single output port at a time, the R5ArbiterN has to select only one packet at a time based on the arbitration policy. The arbiter transmits the selected packet to the router R2. The transmitted packet is stored in the South input port buffer of router R2. This buffer is named as R2BufferS as shown in Fig. 2. The route computation for the packet is performed at the R2SwitchS. As per the position of router R2 in the reference Fig. 1, there are three possible directions where the packet can move. These directions are towards the East port or towards the West port or towards the Local port. Based on the destination address of the packet, R2SwitchS determines the output port and requests the corresponding arbiter for transmission. If R2 is the destination router for the packet, it has already reached the destination router. In such a case, the packet would be directed to the arbiter at the local port and eventually delivered to the local core. If R2 is not the destination router, the packet will move to the adjacent router in the East or West direction based on the routing decisions.

B. SYNCHRONIZATION BETWEEN NoC COMPONENTS

Current buffer storage in a router must be cleared after a packet has moved to another adjacent router. The switch and the arbiter also get ready for processing of a new packet only after the transmission of the current packet is completed. For the lossless packet transmission and correct functioning in an NoC, it is essential to maintain synchronization within a router between buffer, switch, arbiter and synchronization between adjacent routers. In each of the input ports two FSMs namely *sync* and *return* are used to maintain the synchronization. The synchronization is maintained with the help of the handshaking principle.

1) SYNCHRONIZATION BETWEEN TWO ROUTERS

The synchronization between two routers is controlled by a *sync* FSM in our FSM based NoC model. A buffer that presents at the input port of an NoC router, accepts packets from an arbiter in the adjacent router. Before transmitting a packet, the arbiter has to ensure that the buffer in the next router is free. Moreover, the packet cannot be deleted from the previous router until it is safely transferred to the next router. The *sync* FSM maintains the synchronization between the arbiter and buffer between two routers as shown in Fig. 2. The detail modeling of a *sync* FSM is described in the Subsection III-C. The high-level overview depicting the interaction of a *sync* FSM with other NoC functional units is shown in Fig. 2. The dotted double-ended arrow connecting R5ArbiterN and R2BufferS via R2SyncS indicates synchronization between two routers.

2) SYNCHRONIZATION WITHIN ROUTER COMPONENTS

The synchronization between NoC components within a router is maintained using a dedicated FSM named *return* in our FSM based NoC model. After transmitting a packet from a router, all FSMs in that router that are associated with that transmission need to return to their respective initial states. The detail model for a *return* FSM is described in the Subsection III-E. The high-level overview depicting the interaction of a *return* FSM with other NoC functional units is shown in Fig. 2. Here, the dotted double-ended arrows connecting R2ReturnS with R2BufferS, R2SwitchS and all the arbiters in router R2 represent the synchronization between components in R2.

The paths for packet movements are shown with the solid blue arrows in Fig. 2. In a five-port router, we have five such synchronization FSMs corresponding to each port. In this work, we present an input buffer router model. The design of an output buffer router model is similar with minor changes in synchronization.

C. MODELING BUFFER USING FSM

Buffer is present at the input port of a router for storing an incoming packet until it is transmitted. Synchronization between two routers must be taken care of before storing a packet into the buffer. The *sync* FSM controls synchronization between a *buffer* and its corresponding adjacent router, which transmits a packet to that buffer. The basic idea here is that the *sync* will read the current state of the buffer and the arbiter in the adjacent router. It allows the transfer of packets only if there is a slot free in the buffer and allows the buffer to change its state. Thus, the flow control between routers is taken care in our model. For convenience, we present the FSM model for *sync* and *buffer* together.

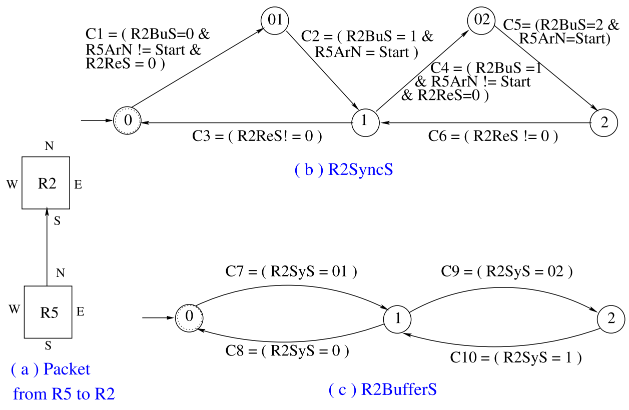
C1 = (R2BuS=0 & R5ArN != Start & R2ReS = 0)
01
C2 = (R2BuS = 1 & R5ArN = Start)
02
C5= (R2BuS=2 & R5ArN=Start)
C4 = (R2BuS =1 & R5ArN != Start & R2ReS=0)
0
1
2
C3 = (R2ReS! = 0)
C6 = (R2ReS != 0)
(b) R2SyncS
N
W
R2
E
S
N
W
R5
E
S
(a) Packet from R5 to R2
C7 = (R2SyS = 01)
C9 = (R2SyS = 02)
C8 = (R2SyS = 0)
C10 = (R2SyS = 1)
(c) R2BufferS

FIGURE 3. **Buffer and sync: (a) Packets from R5 to R2, (b) R2SyncS: Synchronizing between R2BufferS and R5ArbiterN, (c) R2BufferS: Buffer at S input port of R2.**

1) FSM MODEL OF SYNC

A *sync* FSM synchronized with the arbiter in the adjacent router. The FSM model for *sync* is shown in Fig. 3(b). We consider the *sync* FSM at the S input port of the R2 router, with reference to the 3×3 Mesh NoC in Fig. 1(a), for explaining the synchronization with buffer at the same input port. Following the naming convention, we name this *sync* FSM as R2SyncS. Here, R2 indicates the associated router name and S indicates the associated port name. The FSM model representing R2SyncS and its transitions are depicted in Fig. 3(b).

If the current state of $R2SyncS = 0/\ 1/\ 2$, it indicates that the R2BufferS stores zero/ one/ two packet(s). The states $R2SyncS = 01/\ 02$ are considered as intermediate transition states that are used by the corresponding buffer to change its state. Initially R2SyncS is in state 0. The condition for the transition from $(0 \rightarrow 01)$ in Fig. 3(b) is $\mathbf{C1} = (\text{R2BuS} = 0 \wedge \text{R2ReS} = 0 \wedge \text{R5ArN} \neq \text{Start})$. Here, $(\text{R2BuS} = 0)$ means buffer is free as no packet is stored in R2BufferS. The second condition $(\text{R5ArN!} = \text{Start})$ means the R5ArbiterN is ready to transmit a packet. The third condition $(\text{R2ReS} = 0)$ means the South input port is ready to accept a packet and transmits to the desired router if required. If all three conditions are satisfied, the transition $(0 \rightarrow 01)$ takes place. The next transition $(01 \rightarrow 1)$ in Fig. 3(b) takes place if the R5ArbiterN of router

R5 has returned to its initial state after transmitting the packet and the R2BufferS also stored the packet and updated its state. This is represented by the condition **C2** = (R2BuS = 1 ∧ R5ArN = Start). Two transitions are possible from the state 1 as shown in Fig. 3(b). If the transmission for the packet is over, i.e. **C3** = (R2ReS ≠ 0), the FSM returns to the initial state (1 → 0). Here, (R2ReS ≠ 0) means transmission of a packet is completed. On the other hand, if another packet has arrived before the transmission is completed, i.e., **C4** = (R2BuS = 1 ∧ R5ArN ≠ Start ∧ R2ReS = 0), the FSM state changes with the transition (1 → 02). Here, (R2BuS = 1) indicates that one packet is stored in the buffer and it can accommodate another packet. We consider a two slot buffer in this example. In similar way, all other transitions in Fig. 3(b) take place.

2) FSM MODEL OF BUFFER

We consider a buffer (R2BufferS) that has the capacity for storing two packets in the presented model. All the transition in a buffer are controlled by the status of the corresponding *Sync* FSM. The transitions in R2BufferS is controlled by the current state of R2SyncS. All transitions for R2BufferS are shown with the FSM model in Fig. 3(c). Here, satisfying a condition **C7** = (R2SyS = 01) means that a packet is transmitted from the adjacent router. R2BufferS stores that packet and changes its state (0 → 1). The a condition **C8** = (R2SyS = 0) indicates that the transmission is over and buffer has to return to initial state. The condition **C3** = (R2SyncS = 02) indicates that another packet has arrived when the buffer is already storing one packet. Therefore, the second packet is accommodated, and the transition (1 → 2) takes place Fig. 3(c). In this way, the state of a *buffer* is controlled by a *sync* FSM.

D. FSM MODEL OF SWITCH

A switch considers the packet stored in the buffer and redirects it to the proper output port for transmission into the adjacent router. We consider the R2SwitchS at S port of router R2 to explain the design. The FSM model is shown in Fig. 4. The switch remains in the initial state *Wt* (Wait) if no packet is stored in the buffer. If a packet is stored in the buffer and the required condition **C1** = (R2BuS ≠ 0 ∧ R2ReS = 0) is also satisfied, the state would be changed to *C* (Wait → C). Here, (R2BuS ≠ 0) in **C1** indicates the buffer is not empty. At least one packet is stored in the buffer, which needs to be transmitted toward its destination. The condition (R2ReS = 0) in **C1** means the input port is ready to work on a new packet for transmission. The state C stands for compute where routing decision is taken.

All possible routing directions of a packet are considered. One routing direction is selected based on the given routing algorithm. At the state *C*, a variable (*Route*) is considered. The *Route* variable takes any values from 0, 1 and 2, which indicates the direction toward L, E and W ports, respectively. We have not considered any specific routing algorithm here. Considering a routing algorithm demands for packet

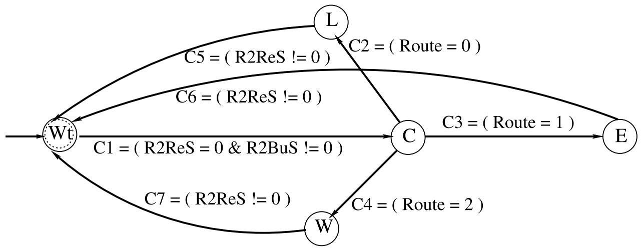

FIGURE 4. **R2SwitchS: The switch at the South port of router R2.**

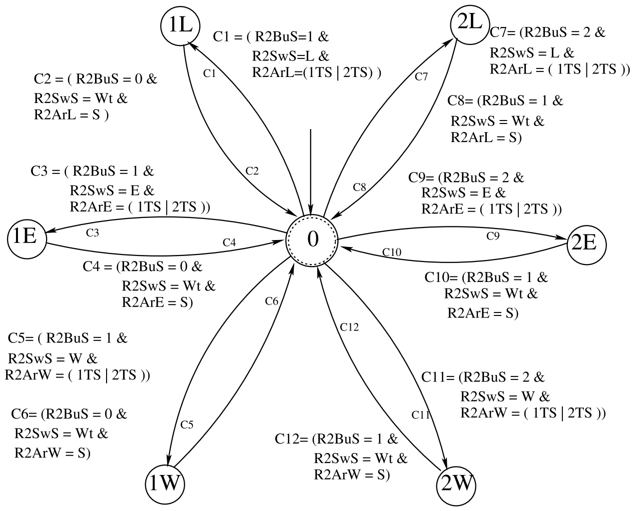
1L
C1 = (R2BuS=1 & R2SwS=L & R2ArL=(1TS | 2TS))
2L
C7= (R2BuS = 2 & R2SwS = L & R2ArL = (1TS | 2TS))
C2 = (R2BuS = 0 & R2SwS = Wt & R2ArL = S)
C8= (R2BuS = 1 & R2SwS = Wt & R2ArL = S)
C3 = (R2BuS = 1 & R2SwS = E & R2ArE = (1TS | 2TS))
C9= (R2BuS = 2 & R2SwS = E & R2ArE = (1TS | 2TS))
1E
0
2E
C4 = (R2BuS = 0 & R2SwS = Wt & R2ArE = S)
C10= (R2BuS = 1 & R2SwS = Wt & R2ArE = S)
C5= (R2BuS = 1 & R2SwS = W & R2ArW = (1TS | 2TS))
C11= (R2BuS = 2 & R2SwS = W & R2ArW = (1TS | 2TS))
C6= (R2BuS = 0 & R2SwS = Wt & R2ArW = S)
C12= (R2BuS = 1 & R2SwS = Wt & R2ArW = S)
1W
2W

FIGURE 5. **R2ReturnS: Synchronization between S port Buffer, S port Switch and Arbiters at L, E and W ports.**

information as well. Instead, all possible paths would be considered in the verification by taking one path at a time. In the simulation approach, the routing direction for a packet is determined by calling a routing algorithm. Since this work does not intend to develop a simulation framework, a specific routing algorithm is not considered. For implementing simulation framework using the proposed FSM model, a routing algorithm is needed to be invoked when the FSM reaches the state *C*. This kind of approach in used in the work [21]. R2SwitchS reaches its appropriate state based on the determined routing direction. After the packet is transmitted, the condition C5 or C6 or C7 becomes True, and the FSM transits back to its initial state *Wt*. Here, **C5** = **C6** = **C7** = (R2ReS ≠ 0). It means the packet is transmitted and all the respective FSMs need to return to their respective initial states. Until then R2ReturnS would not changes its state and the condition (R2ReS ≠ 0) remains True.

E. FSM MODEL OF RETURN

Once the transmission for a packet is over, the *buffer*, *switch* and *arbiter* that are involved in that transmission return to their respective initial states based on the status of the *return* FSM. When the *return* FSM is in a state ≠ 0, it means the transmission for the packet is over. Therefore, all the associated FSMs have to return to their initial states. The *return* FSM transits to the initial state 0 only after all FSMs related to that transmission return to their respective initial states. All the transitions for the *return* FSM R2ReturnS is presented in Fig. 5.

Initially, the R2ReturnS remains in the state 0. If the buffer R2BufferS contains one packet and that packet is transmitted via the E output port, and the condition C3 is satisfied, the state of the R2ReturnS changes to $1E$. Similarly, in case the buffer contains two packets and one packet is transmitted through the E output port and the condition C9 is satisfied, the R2ReturnS changes its state to $2E$. When **C3** $=$ ((R2BuS $=$ 1) $\wedge$ (R2SwS $=$ East) $\wedge$ (R2ArE $=$ (1TS $\vee$ 2TS)) is satisfied the transition (0 $\rightarrow$ 1E) takes place. The meaning of **C3** is that the packet at R2BuS wants to move towards the East output port and it has won the arbitration at R2ArE. Here, (R2BuS $=$ 1) indicates that one packet is stored in the buffer, (R2SwS $=$ East) indicates that the packet wants to move toward the East output port, and (R2ArE $=$ (1TS $\vee$ 2TS) means the South input port packet has won the arbitration for the East output port. If the condition **C4** $=$ ((R2BuS $=$ 0) $\wedge$ (R2SwS $=$ Wait) $\wedge$ (R2ArE $=$ Start)) is satisfied the transition (1E $\rightarrow$ 0) takes place. The meaning for (R2BuS $=$ 0) is the buffer return to initial state, (R2SwS $=$ Wait) means R2SwitchS return to its initial states and (R2ArE $=$ Start) means R2ArbiterE returns to its initial state. Therefore, if **C4** is satisfied the transition (1E $\rightarrow$ 0) takes place, i.e., R2ReturnS returns to initial state. The condition C4 is satisfied only when all FSMs associated with this transmission return to their initial states. After that R2ReturnS also transits back to initial the state. For the condition **C9** buffer is considered as (R2BuS $=$ 2). Other conditions are same as that of the condition **C3**. Similarly, for the condition **C10**, the sate of the buffer becomes (R2BuS $=$ 1) from the state (R2BuS $=$ 2). Other conditions are same as that of the condition **C4**. In a similar way, all other transitions for R2ReturnS take place, as shown in Fig. 5.

F. APPROACH FOR DESIGNING VIRTUAL CHANNELS

Blocking problem of one packet by another packet is resolved with the help of virtual channels (VCs) where a set of VCs share the same physical channel [21], [22]. In VC, a *buffer* is restructured into separate smaller *buffers*. Corresponding to each VC there is a need for separate *buffer* (smaller *buffers*) with corresponding *sync*, *switch* and *return* FSM. An *arbiter* resolves conflict for a physical channel at an output port which is shared by a set of VCs. If the number of VCs increases, the states in an *arbiter* also increase. For the sake of simplicity, in this work, we present *arbiter* with a single VC only.

G. FSM MODEL OF AN ARBITER

Packets from more than one input port might compete for a single output port at a time. These conflicts are resolved by an arbiter with the help of an arbitration policy like fixed-priority, round-robin, first-come-first-serve, weighted round-robin, etc.. We have designed fixed-priority arbiter and round-robin arbiter in this work.

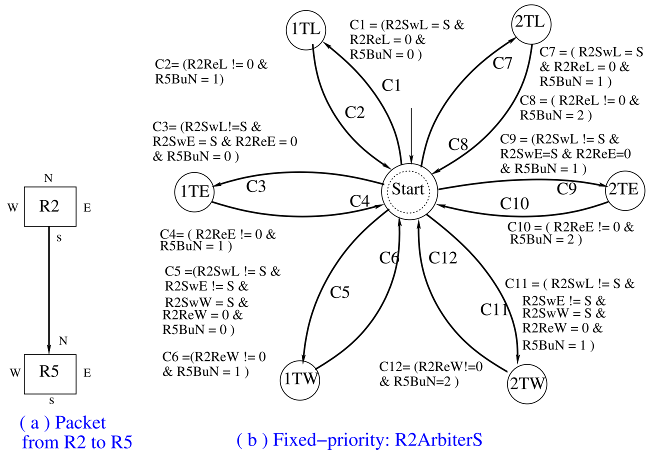
N
W
R2
E
S
N
W
R5
E
S
(a) Packet from R2 to R5
1TL
2TL
1TE
Start
2TE
1TW
2TW
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C1 = (R2SwL = S & R2ReL = 0 & R5BuN = 0)
C2= (R2ReL != 0 & R5BuN = 1)
C3= (R2SwL!=S & R2SwE = S & R2ReE = 0 & R5BuN = 0)
C4= (R2ReE != 0 & R5BuN = 1)
C5 =(R2SwL != S & R2SwE != S & R2SwW = S & R2ReW = 0 & R5BuN = 0)
C6 =(R2ReW != 0 & R5BuN = 1)
C7 = (R2SwL = S & R2ReL = 0 & R5BuN = 1)
C8 = (R2ReL != 0 & R5BuN = 2)
C9 = (R2SwL != S & R2SwE=S & R2ReE=0 & R5BuN = 1)
C10 = (R2ReE != 0 & R5BuN = 2)
C11 = (R2SwL != S & R2SwE != S & R2SwW = S & R2ReW = 0 & R5BuN = 1)
C12= (R2ReW!=0 & R5BuN=2)
(b) Fixed−priority: R2ArbiterS

FIGURE 6. **Fixed-priority arbiter (R2ArbiterS) at the South port of R2 router.**

1) FSM MODEL OF FIXED-PRIORITY ARBITER

The priorities in a fixed priority arbiter are predetermined and we have considered $>$ East $>$ West $>$ North $>$ South as the fixed priority order. It means, the L input port packets have the highest priority and the S input port packets have the lowest priority. A fixed-priority arbiter R2ArbiterS is shown using FSM in Fig. 6. The arbiter transmits packets to the buffer R5BufferN that is present in its adjacent router R5. The initial state of R2ArbiterS is *Start*. A packet from the L input port wins arbitration at R2ArbiterS if the adjacent router buffer R5BufferN is empty and condition C1 is satisfied. In that case, the arbiter state changes to *1TL* from the initial state *Start*. Similarly, if the adjacent buffer has already stored one packet and the condition C7 is satisfied, then also the L input port wins the arbitration. Here the condition **C1** $=$ ((R5BuN $=$ 0) $\wedge$ (R2ReL $=$ 0) $\wedge$ (R2SwL $=$ S)). The condition (R2SwL $=$ S) indicates a packet at the Local input port needs to move towards the South output port. The condition (R2ReL $=$ 0) indicates the readiness of the Local input port for new packet and the (R5BuN $=$ 0) indicates the R5BufferN is free to receive packets as it is not storing any packet. If **C1** is satisfied, the transition (Start $\rightarrow$ 1TL) takes place in Fig. 6. It means that the R5BufferN in the next router is free, and the Local input port wins the arbitration. After receiving the packet by the R5BufferN and once the condition C2 is satisfied, the R2ArbiterS transits to the initial state *Start*. Here, the condition **C2** $=$ ((R2ReL $\neq$ 0) $\wedge$ (R5BuN $=$ 1)). The condition (R5BuN $=$ 1) indicates the packet is transferred to the next router R5. The condition (R2ReL $\neq$ 0) indicates that all the FSMs involved have to return to their respective initial states as the transmission is over. The transitions due to condition C7 and C8 takes place in a similar way. If one packet is already stored in the buffer and another packet is to be stored, in such cases, the transitions take place with the help of the condition C7 and C8. All the other transitions for R2ArbiterS are carried out in a similar way as shown in the Fig. 6.

2) FSM MODEL OF ROUND-ROBIN ARBITER

The priorities in a round-robin arbiter are not fixed and they are getting updated dynamically. When a packet is transferred from an input port, the priority for that input port is set to be the lowest so that other input port packets also get a chance for transmission. To keep the design simple, we have designed round-robin priority generator and round-robin arbiter in two separate FSMs.

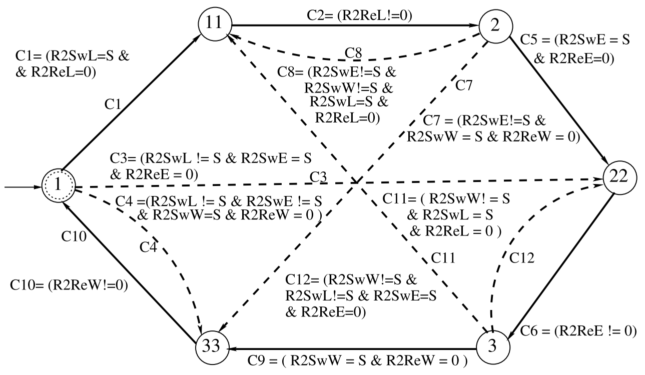
11
2
22
3
33
1
C1= (R2SwL=S & & R2ReL=0)
C1
C2= (R2ReL!=0)
C5 = (R2SwE = S & R2ReE=0)
C8
C8= (R2SwE!=S & R2SwW!=S & R2SwL=S & R2ReL=0)
C7
C7 = (R2SwE!=S & R2SwW = S & R2ReW = 0)
C3= (R2SwL != S & R2SwE = S & R2ReE = 0)
C3
C4 =(R2SwL != S & R2SwE != S & R2SwW=S & R2ReW = 0)
C4
C11= (R2SwW! = S & R2SwL = S & R2ReL = 0)
C11
C12
C10
C10= (R2ReW!=0)
C12= (R2SwW!=S & R2SwL!=S & R2SwE=S & R2ReE=0)
C6 = (R2ReE != 0)
C9 = (R2SwW = S & R2ReW = 0)

FIGURE 7. Round-robin priority generator (R2PriorityS) at the S output port of router R2.

Round-robin priority generator: A round-robin priority generator is shown in Fig. 7. Each state represents a priority value. Initially, it is in the initial state *1*. The priority value *1* indicates the next preferable port to win arbitration is L, *2* indicates the next preferable port is E and so on. If the current state of R2PriorityS is *1* and a packet is destined towards the S output port from the L input port, the state of the priority generator changes to *11*. The priority *11* indicates that L input port packet gets selected for transmission. Similarly, the priority values *22* or *33* indicate that the packet present at the E or W input port, respectively, wins the arbitration. Let the current state of R2PriorityS in Fig. 7 is *1* and no packet is competing for the S output port from the L input port. If a packet competes from the E input port and the C3 condition becomes True then the East input port wins arbitration. In that case the priority changes to state *22* from the previous state *1*. Here, the condition **C3** = ((R2SwE = S) $\wedge$ (R2ReE = 0) $\wedge$ (R2SwL $\neq$ S)). Here, (R2SwL $\neq$ S) means that no packet is competing from the L input port towards the S output port and the (R2SwE = S) means that a packet from the E input port wants to transmit via the S output port. The East input port is also ready for a new packet if the condition (R2ReE = 0) is True. If the overall **C3** is satisfied, the transition (1 $\rightarrow$ 22) takes place. It means a packet wants to traverse from the E input port to the S output port, and there is no competing packet from the L input port to the same output port. Therefore, priority changes to 22 in favour of the East port. Similarly, if there are no packets from the Local and East input ports towards the South output port and at the same time, a packet is destined from the West input port towards the South output port, then the condition C4 is satisfied. In that case, the priority would change to 33 in favour of the West input port. The other transitions in the priority generator of Fig. 7 can be explained in the similar fashion.

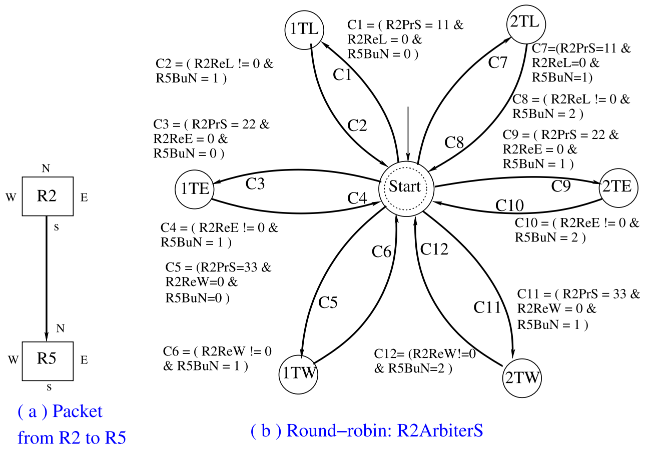
N
W
R2
E
S
N
W
R5
E
S
(a) Packet from R2 to R5
1TL
2TL
1TE
Start
2TE
1TW
2TW
C1 = (R2PrS = 11 & R2ReL = 0 & R5BuN = 0)
C1
C2 = (R2ReL != 0 & R5BuN = 1)
C2
C7
C7=(R2PrS=11 & R2ReL=0 & R5BuN=1)
C8 = (R2ReL != 0 & R5BuN = 2)
C8
C3 = (R2PrS = 22 & R2ReE = 0 & R5BuN = 0)
C3
C4
C9 = (R2PrS = 22 & R2ReE = 0 & R5BuN = 1)
C9
C10
C10 = (R2ReE != 0 & R5BuN = 2)
C4 = (R2ReE != 0 & R5BuN = 1)
C5 = (R2PrS=33 & R2ReW=0 & R5BuN=0)
C5
C6
C12
C11
C11 = (R2PrS = 33 & R2ReW = 0 & R5BuN = 1)
C6 = (R2ReW != 0 & R5BuN = 1)
C12= (R2ReW!=0 & R5BuN=2)
(b) Round–robin: R2ArbiterS

FIGURE 8. Round-robin arbiter R2ArbiterS at the South port of Router R2.

Round-robin Arbiter: The round-robin arbiter uses the priority set by the corresponding round-robin priority generator. Fig. 8(b) shows the FSM representation of the round-robin arbiter R2ArbiterS along with its transitions. In round-robin arbiter, we use priority information generated by a priority generator as shown in Fig. 7. The priority values (*11, 22, 33*) indicate the packet from the input port that wins the arbitration. Therefore, the state of the *switch* FSMs are not explicitly considered in the transitions for round-robin arbiter as they are considered during determining the priority. The FSMs for round-robin arbiter is similar with the FSM for fixed-priority arbiter in the remaining other aspects. If the priority is set to *11*, a packet from the Local input port would be transmitted when buffer in the next router is free (condition C1 or C7 in Fig. 8). The next state of the R2ArbiterS changes to *1TL* or *2TL*. Here, the condition **C1** = ((R5BuN = 0) $\wedge$ (R2ReL = 0) $\wedge$ (R2PrS = 11)). The condition (R2PrS = 11) indicates that the priority is set for the Local input port, (R2ReL = 0) means the Local input port is ready and (R5BuN = 0) indicates that the corresponding buffer in the adjacent R5 router is free. Therefore, the transition (Start $\rightarrow$ 1TL) takes place with an indication that the Local input port wins the arbitration. On satisfying the condition **C2** = ((R5BuN = 1) $\wedge$ (R2ReL $\neq$ 0)), the transition (1TL $\rightarrow$ Start) takes place. It means the R2ArbiterS returns to its initial state once the transmission from the Local port is over. Here, the condition (R5BuN = 1) means that the buffer in the adjacent router R5 is updated from (R5BuN = 0) after receiving the packet. The condition (R2ReL $\neq$ 0) means all the FSMs corresponding to this transmission has to return to their respective initial states. The other condition regarding the transfer of packet from the local port **C7** = (R5BuN = 1 $\wedge$ R2ReL = 0 $\wedge$ R2PrS = 11) is similar to **C1**. The only difference is, in case of **C1** the adjacent router buffer was empty and in case of **C7** the adjacent router buffer has stored a packet and it has the capacity to store another packet. On satisfying **C7**, the transition (Start → 2TL) takes place. Here, the state 2TL means R5BufferN is storing one packet and has capacity for one more packet. The state 1TL means R5BufferN is empty at present. In similar way, all other transitions in the round-robin arbiter in Fig. 8 are carried out.

IV. CORRECTNESS OF THE MODEL

We have verified progress and synchronization between functional units for ensuring the correctness of our FSM based NoC model. Progress in a communication network ensures liveness of the system [3]. That means a system component should not stuck in a state and each component of the system is functioning. Satisfying progress in the proposed NoC model implies that current state of an FSM does not stuck permanently in a state, i.e., the state of an FSM keeps on changing provided that there presents a packet as input for transmission. In NoC context, simply satisfying progress property locally does not guarantee deadlock freedom. Satisfying progress ensures only the correctness of the system. For deadlock-freedom, global deadlock needs to be avoided in consideration with a specific routing algorithm.

A. PROGRESS IN ROUTER COMPONENTS

In this subsection we have presented the LTL specifications for the progress in the router components: *buffer*, *switch* and *arbiter*.

1) PROGRESS IN A BUFFER

The change of states in a *buffer* is controlled by the state of the *sync* FSM. Initially both the *buffer* and *sync* FSMs are at their initial state 0. If the *sync* FSM changes its state to 01 with the transition (0 → 01) in Fig. 3(b), the corresponding *buffer* also needs to change its state with the transition (0 → 1) in Fig. 3(c) in the next cycle. Formally this property is written as, "If the current state of *sync* FSM is 01, the next state of the corresponding *buffer* changes to 1 in the next cycle". The Linear Temporal Logic (LTL) for this property is, `G((R2SyncS = 01) ⟹ X(R2BufferS = 1))`. Similarly, LTL specification for the progress property when another packet is stored in the buffer is, `G((R2SyncS = 02) ⟹ X(R2BufferS = 2))`. Satisfying these properties ensures the correctness of the buffer model.

2) PROGRESS IN A SWITCH

If there present at least one packet in a buffer, the corresponding switch should compute the routing direction for transmitting the packet. Formally this property is written as, "If the input buffer is non-empty and the switch is currently in waiting(Wt) state, the switch state will eventually change to compute(C) state". The specification using LTL for this property is, `G(((R2BufferS != 0) ∧ (R2SwitchS = Wt) ⟹ F((R2SwitchS = C))`. Similarly, "If switch is currently in compute(C) state, the switch state will eventually change to L or E or W as per the routing decision". The LTL representation is, `G((R2SwitchS = C) ⟹ F(R2SwitchS = L ∨ R2SwitchS = E ∨ R2SwitchS = W))`.

3) PROGRESS IN A FIXED-PRIORITY AND ROUND-ROBIN ARBITERS

When packets from input ports intend to get transmitted via a particular output port, the corresponding arbiter in that output port must decide in favour of an input port packet for transmission. Formally this property is written as, "If an arbiter is in initial state (Start) and there is at least one packet requesting that output port, the state of the arbiter will eventually change". The LTL representation is, `G((R2ArbiterS = Start) ∧ ((R2SwitchL = S) ∨ (R2SwitchE = S) ∨ (R2SwitchW = S)) ⟹ F(R2ArbiterS != Start))`. In both the fixed-priority arbiter and round-robin arbiter, the progress does not ensure the starvation freedom. Since states are same for the fixed-priority arbiter in Fig. 6 and the round-robin arbiter in Fig. 8, same LTL specification is used for both the arbiters.

B. SYNCHRONIZATION WITHIN A ROUTER

The synchronization within a router is controlled by a *return* FSM present at each input port. Initial state of a *return* FSM is 0. If (*Return*! = 0), it implies that a packet is chosen for transmission and the corresponding FSMs associated with that transmission have to return to their respective initial states. Once these FSMs return to their initial states, the state of *return* FSM too changes to (*Return* = 0). A new transmission from that port starts only after that point. Correct execution of these transitions in order indicates proper synchronization inside a router. One synchronization property is, "When an input port packet is selected for transmission by an arbiter, the state of the *return* FSM corresponding to that input port changes from its initial state". Its LTL representation is, `G((R2ReturnE = 0) ∧ (R2BufferE = 1) ∧ (R2SwitchE = S) ∧ (R2ArbiterS = (1TE ∨ 2TE)) ⟹ X(R2ReturnE = 1S))`. Once the transmission from the East input port via the South output port in R2 router is over, the East input port is not ready for another packet until the corresponding *buffer*, *switch* and *arbiter* return to initial states. When these FSMs return to initial states, the *return* FSM too returns back to initial state (*R2ReturnE* = 0) (Fig. 5). Processing of new packets are enabled only when (*R2ReturnE* = 0). This synchronization property can be expressed as, "After the transmission of a packet that is stored in an input port buffer if the corresponding buffer sets its buffer slot as free, corresponding switch and arbiter return to their initial states then the corresponding *return* FSM too returns to its initial state". In LTL, `G(((R2ReturnE = 1S) ∧ (R2BufferE = 0) ∧ (R2SwitchE = Wt) ∧ (R2ArbiterS = Start) ⟹ X((R2ReturnE = 0))`. In similar way, synchronization between FSMs within a router for all ports can be represented using LTL properties for verification.

C. CORRECTNESS OF A PRIORITY GENERATOR

The priority generated by a priority generator for a round-robin arbiter has to be checked if the priorities are generated correctly. Since no priority is used for the fixed-priority arbiter we have not checked this for the fixed-priority arbiter.

For proper functioning of the round-robin arbiter, the priority needs to be set eventually for each input port by the corresponding round-robin priority generator in Fig. 7. For example, in router R2, packets from L input port (priority *1*, *11*), E input port (priority *2*, *22*) and W input port (priority *3*, *33*) compete for S output port. If current state of the priority generator is *1* and a packet from the West input port competes for the South output port, eventually the priority should be set to *33* so that a packet from the West input port gets preference for being transmitted. This is represented in LTL as, $\texttt{G((R2PriorityS = 1)} \wedge \texttt{(R2SwitchW = S)} \wedge \texttt{(R2ReturnW = 0)} \implies \texttt{F(R2PriorityS = 33))}$. Satisfying this LTL specification indicates correct functioning of the round-robin priority generator. Similarly, in router R5, if a packet from the South input port competes for the North output port, the priority for the South input port must set to *44* eventually. To verify this, its LTL specification is represented as, $\texttt{G((R5PriorityN = 1)} \wedge \texttt{(R5SwitchS = N)} \wedge \texttt{(R5ReturnS = 0)} \implies \texttt{F(R5PriorityN = 44))}$. Verification results for all such LTL specifications must be True for assuring the correctness in the design of round-robin priority generator.

V. APPLICATION OF THE MODEL

As applications of the presented FSM based NoC models, verification of starvation-freedom and transfer of packets across routers are presented in this section. We also discuss the challenges of extensive state space in a complete NoC and our approach to verify overall NoC considering the NoC in part-by-part.

A. VERIFICATION OF STARVATION-FREEDOM

Starvation-freedom is defined as fairness in resource allocation between competing agents. Starvation at an output port of an NoC depends upon the underlying arbitration logic, the fixed-priority and the round-robin arbiters for this work. There may be more than one packet from other input ports that all want to exit through the same output port. If only one input port packet keeps on getting preference, the packets from other ports have to wait indefinitely and they suffer from starvation. Starvation-freedom in NoC context can be expressed as, "If a packet from an input port competes for an output port, eventually the output port arbiter has to select that input port for transmitting the packet". For example, "If a packet from the West input port intends to exit through the South output port in R2 router, the packet should get a chance to exit through the South output port in future." The LTL representation of the same is, $\texttt{G((R2SwitchW = S)} \wedge \texttt{(R2ArbiterS = start)} \implies \texttt{F((R2ArbiterS = 1TW)} \vee \texttt{(R2ArbiterS = 2TW))}$. By satisfying this property, it implies that every packets from the West input port intending for the South output port will be transmitted in future. If this property becomes False, it implies that for some possible scenarios, packets from the West input port to the South output port never get a chance for the transmission. The R1ArbiterE, in the R1 router of Fig. 1, accepts packets from the Local and the South input ports. LTL representation of starvation-freedom at the R1ArbiterE for the South input port is, $\texttt{G(((R1SwitchS = E)} \wedge \texttt{(R1ArbiterE = Start))} \implies \texttt{F((R1ArbiterE = 1TS)} \vee \texttt{(R1ArbiterE = 2TS))}$. Meaning of this LTL is, globally if the current state of R1ArbiterE is *Start* and R1SwitchS is *E*, eventually the R1ArbiterE would transit to *1TS* or *2TS*. Both states *1TS* or *2TS* in R1ArbiterE imply that a packet from the South input port is selected by the arbiter. If the mentioned LTL is satisfied, the starvation-freedom for the South input port is ensured at the R1ArbiterE. Similarly, the starvation-freedom specification for any input port at a given output port is represented using LTL.

B. VERIFICATION FOR A TRANSFER OF PACKET

The correct synchronization between two routers implies the transfer of packets between two routers in a proper way. A desirable property for a buffer is, "If a buffer is empty and the corresponding arbiter in the adjacent router has selected a packet for transmission, the state of the buffer changes eventually to store the packet". The LTL specification for the same is, $\texttt{G((R5ArbiterN != Start)} \wedge \texttt{(R2BufferS = 0)} \wedge \texttt{(R2ReturnS = 0)} \implies \texttt{F(R2BufferS = 1))}$. The same property can be expressed as, "An arbiter is ready for transmitting a packet and the connected buffer in the adjacent router is free for accommodating a packet, eventually the packet is stored in the next router buffer". Thus, the LTL representation for transfer of packets between two routers are presented for each pairs of connected routers in an NoC.

C. VERIFICATION OF OVERALL NoC

Considering all the routers in a complete NoC along with the respective router components results in an extensive state space. In this section, we discuss the state space for our FSM based NoC model by considering the number of FSMs needed for the complete NoC model. We have also presented the active window concept where a part of NoC is considered instead of considering the complete NoC at a time.

1) NUMBER OF FSMs IN AN NoC

The number of active ports in an NoC router depends upon its position. All ports may not be active in an NoC router. Considering the 3×3 Mesh NoC in Fig. 1(a), for all the corner routers namely R1, R3, R7 and R9 has three ports active. Two ports connect its two neighbour and another port is for its local core. There present 4 such corner routers in a Mesh NoC of any size. The router R2 in Fig. 1(a) has three neighbours. Therefore, router R2 has total four active ports, three ports connecting each neighbour and another port connecting the

TABLE 2. Number of FSMs in NoCs considering Fixed-priority (FP) and Round-robin (RR) arbiter.

NoC	Number of FSMs in Fixed-priority (FP) arbiter			Total
	Active ports = 3	**Active ports = 4**	**Active ports = 5**	**FSMs**
	$a = (R_3 \times \mathbf{FP}_3)$	$b = (R_4 \times \mathbf{FP}_4)$	$c = (R_5 \times \mathbf{FP}_5)$	$(a+b+c)$
2x2	$4 \times 15 = 60$	$0 \times 20 = 0$	$0 \times 25 = 0$	60
3x3	$4 \times 15 = 60$	$4 \times 20 = 80$	$1 \times 25 = 25$	165
4x4	$4 \times 15 = 60$	$8 \times 20 = 160$	$4 \times 25 = 100$	320
5x5	$4 \times 15 = 60$	$12 \times 20 = 240$	$9 \times 25 = 225$	525
6x6	$4 \times 15 = 60$	$16 \times 20 = 320$	$16 \times 25 = 400$	780
7x7	$4 \times 15 = 60$	$20 \times 20 = 400$	$25 \times 25 = 625$	1085
8x8	$4 \times 15 = 60$	$24 \times 20 = 480$	$36 \times 25 = 900$	1440
NoC	**Number of FSMs in Round-robin (RR) arbiter**			**Total**
	Active ports = 3	**Active ports = 4**	**Active ports = 5**	**FSMs**
	$a = (R_3 \times \mathbf{RR}_3)$	$b = (R_4 \times \mathbf{RR}_4)$	$c = (R_5 \times \mathbf{RR}_5)$	$(a+b+c)$
2x2	$4 \times 18 = 72$	$0 \times 24 = 0$	$0 \times 30 = 0$	72
3x3	$4 \times 18 = 72$	$4 \times 24 = 96$	$1 \times 30 = 30$	198
4x4	$4 \times 18 = 72$	$8 \times 24 = 192$	$4 \times 30 = 120$	384
5x5	$4 \times 18 = 72$	$12 \times 24 = 288$	$9 \times 30 = 270$	630
6x6	$4 \times 18 = 72$	$16 \times 24 = 384$	$16 \times 30 = 480$	936
7x7	$4 * 18 = 72$	$20 * 24 = 480$	$25 * 30 = 750$	1302
8x8	$4 * 18 = 72$	$24 * 24 = 576$	$36 * 30 = 1080$	1728

local core. We get (N-2)*4 such routers having four active ports in an NxN Mesh NoC. The router R5 in Fig. 1(a) has four neighbours. Therefore, all the five ports are active for router R5. There are (N-2)*(N-2) such routers with five active ports in an N*N Mesh NoC.

In our FSM model for a router with fixed-priority arbiter we need 5 FSMs corresponding to each port of that router. These five FSMs are namely switch, buffer, sync, return and fixed-priority arbiter. If a router with fixed-priority arbiter has three active port, we need 15 ($3 * 5 = 15$) such FSMs to model the router. Therefore, we consider $FP_3 = 15$ in Table 2. Similarly, for a router having fixed-priority arbiter with 4 active ports, $FP_4 = 20$ ($4 * 5$) and with 5 active ports, $FP_5 = 25$ ($5 * 5$). These values are used in Table 2 for calculating the number of FSMs in a Mesh NoC model. If we consider round-robin arbiter in the router, one additional FSM is needed for priority generation. Therefore, for a router with round-robin arbiter we need 6 FSMs corresponding to each port of that router. If a router with round-robin arbiter has three active port, we need 18 ($3*6 = 18$) such FSMs to model the router. Therefore, we consider $RR_3 = 18$ in Table 2. Similarly, for a router having round-robin arbiter with 4 active ports, $RR_4 = 24$ ($4 * 6$) and with 5 active ports, $RR_5 = 30$ ($5 * 6$). The total number of FSMs needed while designing a complete NoC using fixed-priority arbiter and round-robin arbiter is shown in Table 2. The number of FSMs needed for modeling a 8×8 Mesh NoC is 1440 considering FP arbiter. This number is even more while considering round-robin arbiter. For 8×8 Mesh NoC we need 1728 FSMs. The number of states in each FSM is multiplied while a complete NoC is encoded using a model checker. It gives the total state space for a complete NoC. State space for a complete NoC is a too high for achieving scalabilty using model checker. State space for even for a 2×2 NoC is $2^{138.24}$ while using fixed-priority arbiter and is $2^{169.2}$ while using round-robin Arbiter. These numbers explode with the increase of NoC sizes. Therefore, we have considered the active windows next for each NoC router instead of considering a complete NoC at a time.

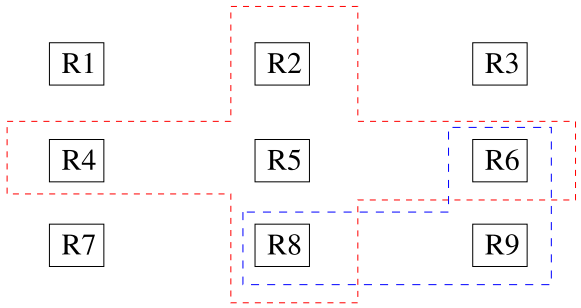

FIGURE 9. Partitioning NoC: Active windows for the Router R5 and R9.

2) ACTIVE WINDOWS

Due to the state space explosion problem, a complete NoC system along with its detailed components cannot be encoded with a state-of-the-art model checker. Therefore, we partition the NoC into active regions and perform verification for starvation and transfer of packets in each active region. Router R9 interacts with its two neighbors R6 and R8. Therefore, we consider router {R6, R8, R9} at a time as the active window for R9, as shown in Fig. 9. Similarly, active window for R5 is {R2, R4, R5, R6, R8} and for R8 is {R5, R7, R8, R9} and so on. The communication between two routers are present in an active window. Therefore, properties involving two routers can be checked using active windows. All these active windows can be executed using parallel threads to save verification time. To verify global properties like deadlock and livelock, we need to consider the complete NoC at a time. Even active window would not be able to facilitate that. Therefore, global properties like deadlock and livelock can not be verified due to state space explosion even by using active windows.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

We have created all FSM models manually from 2×2 to 8×8 Mesh NoC. The FSM models are encoded in NuSMV [9] for verification. Individually executing each router for verification of progress, synchronization, proper priority generation for round-robin arbiter, transfer of packets and starvation are time consuming operations. In this section, experimental results with their analysis are presented for all the experiments. In all the experiments, a machine having Intel Xenon(R) 2.10GHz X 32 processors with 64 GB RAM is used.

TABLE 3. Verification time (H:M:S) for verification of progress, synchronization and priority considering fixed-priority (FP) arbiter (A) and round-robin (RR) arbiter.

Mesh	Serial execution		Parallel execution		Seep Up	
NoC	FP A.	RR A.	FP A.	RR A.	FP A.	RR A.
2x2	0.0.2	0.0.8	0.0.2	0.0.7	1x	1.1x
3x3	1.20.02	3.38.16	1.10.09	2.26.34	1.1x	1.3x
4x4	4.0.17	11.36.23	1.17.09	2.35.16	3.1x	4.5x
5x5	8.02.11	23.54.51	1.26.07	2.41.02	5.6x	8.9x
6x6	13.20.31	40.32.56	1.36.27	2.58.21	8.3x	13.7x
7x7	20.12.03	61.30.42	2.05.03	3.57.06	9.6x	15.6x
8x8	28.02.16	86.48.02	2.38.56	4.49.27	10.6x	18.0x

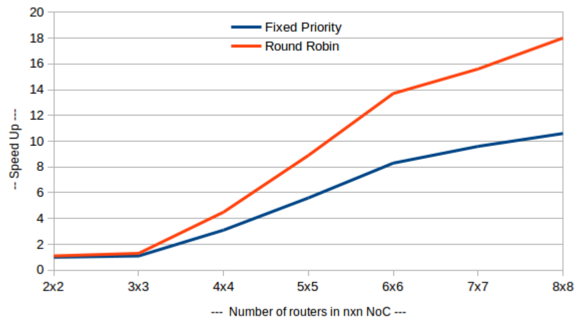

FIGURE 10. Speed up with the increase in the number of routers.

A. VERIFICATION OF PROGRESS, SYNCHRONIZATION AND PRIORITY GENERATION WITHIN A ROUTER

In this experiment, we have used all routers in an NoC individually and have executed them both serially and in parallel threads. We have verified the progress in NoC router components, synchronization within a router and correctness in priority generation using the LTL specifications as described in Section IV. Experimental results show that synchronization properties within a router and progress within a router are satisfied to be True for all routers. The LTL specification for priority generation for a round-robin arbiter also satisfied in all the experiments.

1) RUNTIME IMPROVEMENT WITH PARALLEL EXECUTION CONSIDERING INDIVIDUAL ROUTER

The execution time for the experiments is shown in Table 3. In comparison to fixed-priority arbiter, state-space increases in round-robin arbiter due to the dynamic priority implementation. For that reason, verification time also increases significantly for NoC containing round-robin arbiter. Verification time is shown in Table 3. The number of routers increases for larger NoC. Therefore, as the NoC grid size increases, verification time also increases. The verification time needed for both the serial and parallel execution along with speed ups are shown in Table 3. The ratio of serial execution time with the parallel execution time is presented as the speed up in this work. The experimental results show that parallel threads help to speed up the verification process up to 18x times over the equivalent serial execution in the case of larger NoCs.

We put the speed up trend for parallel execution of individual router in Fig. 10. Though, theoretically we could expect the speedup proportional to the router numbers i.e., n^2 for an nxn NoC, practically we are getting lesser than that. Even the speedup trend is not same for both the Fixed Priority and Round Robin arbiter. Besides the number of routers, other factors like thread overhead, operating system details,and memory requirements for the verification also affect the speed up. The speed up trend continues until the memory is not saturated.

B. VERIFICATION OF TRANSFER OF PACKETS AND STARVATION FREEDOM CONSIDERING ACTIVE WINDOWS

For verifying the transfer of packets, we need to consider an arbiter and the corresponding buffer in the adjacent router, i.e., the connected port of adjacent router. Similarly, for verification of starvation, we need to consider the arbiter, that communicates with the buffer of adjacent router. For considering the communication with an adjacent router, we have considered active windows in this experiment. Examples for active windows are shown in Fig. 9. We have used active window corresponding to each router in an NoC. All active windows are executed using parallel threads to save the verification time.

Successful transfer of packets indicates correct synchronization between an arbiter and a buffer in the adjacent router as well. The LTL specifications that are used for the verification of the transfer of packets are presented in Section V. All the specifications for transfer of packets corresponding to each pair of arbiter and connected buffer are verified to be True in all the experiments. We have verified starvation freedom as well in this experiment. The verification results of starvation-freedom for router R5 in a 3×3 Mesh NoC (Fig. 1) are shown in Table. 4. We get similar results for all the other routers up to 8×8 Mesh NoC. Some ports are inactive (W and N ports in R1 of Fig. 1) and starvation-freedom specifications are not applicable (NA) for some other ports. Such ports in router R5 with respect to different arbiters are denoted as "not applicable (NA)" in Table. 4. In router R5 (Fig. 1), packets stored in Local, West, North and South input ports compete for the East output port (R5ArbiterE). The East input port packets do not compete for the East output port. Therefore, starvation-freedom at R5ArbiterE from the East input port is not applicable and is denoted as NA in Table 4. For the Local, West, North and South input ports, if starvation-freedom is satisfied, it is denoted by T (True). Otherwise, it is denoted by F (False).

1) ANALYSIS OF THE FINDINGS ON STARVATION FREEDOM

For fixed-priority arbiter, it seems starvation-freedom is satisfied only for the highest priority port. Whereas, experimentally it is found that *starvation-freedom specification is satisfied for both the highest and the second highest priority ports.* We consider Local $>$ East $>$ West $>$ North $>$ South as the priority order in this work. Here, Local input port has the highest priority and the South input port has the lowest

TABLE 4. Starvation-freedom: Fixed-priority (FP) arbiter and round-robin (RR) arbiter.

Packets from input ports	Starvation-freedom at the output port of an Arbiter									
	R5ArL		R5ArE		R5ArW		R5ArN		R5ArS	
	FP	RR	FP	RR	FP	RR	FP	RR	FP	RR
L	NA	NA	T	T	T	T	T	T	T	T
E	T	T	NA	NA	T	T	T	T	T	T
W	T	T	T	T	NA	NA	F	F	F	T
N	F	T	F	T	F	T	NA	NA	F	T
S	F	T	F	T	F	T	F	F	NA	NA

TABLE 5. Verification time (H:M:S) for transfer of packets and starvation freedom considering Fixed-priority arbiter (FP A.) and round-robin arbiter (RR A.) with active windows.

NoC	Verification time considering FP and RR Arbiter					
	Serial execution		Parallel execution		Seep Up	
	FP A.	RR A.	FP A.	RR A.	FP A.	RR A.
2x2	0.0.4	0.0.56	0.0.2	0.0.24	2x	2.3x
3x3	7.33.04	70.02.56	6.23.09	38.03.21	1.2x	1.8x
4x4	26.12.04	234.08.56	10.01.13	53.50.09	2.6x	4.4x
5x5	57.57.04	492.18.56	10.21.05	90.45.12	5.6x	5.4x
6x6	102.08.04	844.32.56	11.09.17	152.03.51	9.1x	5.5x
7x7	158.45.04	1290.50.56	13.10.03	226.01.07	12x	5.7x
8x8	227.48.04	1836.57.56	16.58.10	315.12.32	13.4x	5.8x

priority. Experimental results are shown in Table. 4. At the North output port of R5 router (R5ArbiterN), the West and the South input ports suffer from starvation. Both the Local and the East input ports are found to be starvation free. The reason behind these findings is analysed from the the debug trace. Debug trace shows that packets from the Local, East, West and South input ports keeps on trying continuously to access the North output port (R5ArbiterN). A packet from the highest priority port i.e., the Local input port is transmitted to the adjacent router. There present other consecutive packets from the Local port. When transmission of a packet is over, there entails some delay in maintaining synchronization with the buffer (clearing of the buffer without data loss) and the switch using the return FSM. On the other hand, the packet from the next highest priority port, i.e., the East input port, is ready and keeps on waiting for the same R5ArbiterN (North output port). Therefore, a packet from the East input port get the next chance for transmission. The synchronization latency in the highest priority port is the reason for satisfaction of starvation-freedom property for the the second-highest priority input port as well in fixed-priority arbiter. If synchronization latency can be nullified in a model, packets only from the highest priority port, i.e., the Local port in this case, get a chance for transmission. We believe that such delay would present in the actual hardware as well unless it is nullified using extra resources like extra buffers. When the transmission of the packet from the East input is over, in the meantime, the synchronization at the Local input port is completed. In the next transmission, the local input port get the preference for transmission again. This process continues. Only the Local and East input port packets get the chance for transmission alternatively. Packets competing from the West and the South input ports might have to wait indefinitely. Starvation-freedom can not be guaranteed for them. On the other hand, round-robin arbiter is a dynamic arbitration policy. The priority FSM in Fig. 7 changes the priority in round-robin fashion. Therefore, starvation-freedom is ensured for all the input ports. Experimental results are shown in Table. 4.

2) RUNTIME IMPROVEMENT WITH PARALLEL EXECUTION FOR THE ACTIVE WINDOWS

The execution time for transfer of packets and starvation verification using active window is shown in Table. 5. Similar to the previous experiments considering individual routers, NoC with a round-robin arbitration policy takes longer time than NoC considering a fixed-priority arbiter. Experimental results are shown in Table 5. Verification time increases for larger NoC. The number of routers is less in smaller NoC. Moreover, corner routers and boundary routers have less number of active ports. For example, the router R1 in Fig. 1 has three active ports (Local, East and South). But for the intermediate router like R5, all five ports are active. Larger NoC has more number of such routers where all ports are active. For this reason, more time is utilised for the verification of larger NoC. *The parallelization of starvation verification is found to be very effective as the parallel execution helps in saving the execution time significantly. As shown in Table 5, executing starvation verification for fixed-priority arbiter with parallel threads helps to speed up the verification process up to 13.4x times over the equivalent serial execution.* This runtime improvement for round-robin arbiter using parallel thread is 5.8x times over serial execution. The improvement is not significant for round-robin arbiter after a 5×5 NoC because the round-robin arbiter demands more state space.

VII. RELATED WORK

In this section, we have presented existing works on formal modeling of NoC using different formalisms. Due to the enormous benefits of the formal models [11], there have been efforts on formal verification of NoC using both model checking and theorem proving. D. Borrione *et al.* target validation of the communication infrastructure of NoC [1]. Generic NoC is based on an abstract view of the communications network of NoC is presented in this work. This work has not considered router components in detail. The generic NoC model is modeled as the composition of key components like routing, scheduling and interfaces. This model includes topologies, routing algorithms and scheduling policies. A finite number of routers are connected to generic NoC communication architecture. A generic formal model for NoC is implemented using ACL2 theorem prover and simulator. ACL2 provides two functionalities, as a theorem prover and as an execution engine in the same modeling environment. Timing information is not considered in that work, which is incorporated in their next work [2]. This work describes an extension work on generic model for NoC [1]. It is implemented in ACL2 theorem prover, which contains the executable logic as well. This model is useful for serving as a formal reference for the validation and simulation of NoC at the initial design phase. The work demonstrated the transmission of messages on generic communication architecture, with an arbitrary network topology and node interfaces, routing algorithm and switching technique. The model considers its main input, as a list of messages that can be injected in the network. Before injecting in the network, these messages are first encoded. Properties like deadlocks and starvation are not considered in this work and it considers only high-level descriptions of NoC while modeling.

Palaniveloo *et al.* present a new formal model for the existing Hermes NoC architecture along with the communication scheme [6]. They propose Heterogeneous Protocol Automata (HPA) as a language for modeling Hermes NoC as an event based transition system. This work verifies reachability of flits using Spin model checker [23]. At the input port of the Hermes Router [24] it models five bi-directional ports and the bounded buffers. It also includes the XY-routing algorithm, wormhole switching, arbitration logic with priority, and a handshake protocol of the communication scheme. With help of PROMELA, the automata model is mapped manually. PROMELA is specification language for the SPIN model checker [25]. It models XY-routing algorithm by writing codes in PROMELA. This work models NoC components like switch, arbitration by defining Heterogeneous Protocol Automata. It verifies reachability of a flit using SPIN model checker. This work has not checked critical issues like starvation, deadlock, livelock etc.

Lan *et al.* [26] proposes a bidirectional channel Network-on-Chip architecture to enhance the performance of on-chip communication. It allows each communication channel between two routers to be dynamically self configured to transmit flits in either direction. Chen *et al.* present a verification approach on bidirectional NoC in [7]. They have verified mutual exclusion and starvation freedom. Formal Modeling of NoC presented in this work considering the proposed Bidirectional NoC design [26]. A channel has three states: free, idle and waits. One end of a channel is configured as a free state, and data are transmitted from that end. Receiving end of a channel is in idle state. To change a state from idle to free, it goes through the intermediate state called wait state. BiNoC design is converted into Extended Time Automata for verification. State Graph Manipulator (SGM) [27] is used as model checking tool in all the experiments. Due to state space explosion, the work can not verify deadlock freedom. The model used in this work considers only one router and has not considered detailed model of the NoC components. The approach may not scale for a complete NoC. In a Ph.D. dissertation on formal verification of fault tolerant NoC [28], process-algebra is used for modeling. For all the experiments in this work a 2×2 NoC model is considered. Scalability is not achieved in this work. Deadlock verification fails due to state space explosion problem.

There is another approach of modeling interconnection network using executable micro architectural specification (xMAS) [29], [30]. A richer set of micro architectural primitives are identified that allow describing the complete system by composition methods. They are useful as a modeling framework for validating existing micro architecture. These are recent works that usages xMAS as modeling framework in their works [3], [4], [31], [32], [33], [34], [35], [36]. Some existing verification works on communication network [3], [4], [37] use xMAS for modeling the system. Ray *et al.* present progress verification on a communication fabric by breaking end-to-end progress property of a virtual channel into localized progress property [3]. This paper considers a virtual channel which is designed using xMAS [3]. This work focuses on progress property. Progress means whenever there is a packet trying to enter the virtual channel, it will pass through the channel. In this work [3], end-to-end progress property is broken down into localized progress properties. Localized progress are more easily provable, and leads to a formal proof of overall progress. The authors conclude that some more studies are needed to apply the same approach for progress verification in NoC.

In [5], we verify progress property on a xMAS based NoC model using NuSMV [9]. Buffer is not considered in that work. Considering buffer, verification of this model using NuSMV does not scale even upto 2×2 NoC. Using this approach, deadlock verification of NoC is found to be infeasible due to state space exposition problem.

Finite State Machine (FSM) is a popular modeling formalism suitable to use with model checker. NuSMV internally composes all the FSMs present in the system model and verifies the truth value for any given specification with respect to that composed model [11]. In verification process, state space increases exponentially in case of a huge system like NoC. This is called state space explosion problem [38]. Besides formal verification purpose, FSM are being used for various other applications by the research community [12], [13], [14], [15], [16], [17], [18], [19]. In this paper, we have modeled the NoC using FSM first to check locally dependent properties like starvation. We have modeled NoC considering detail components. Synchronization between NoC components is essential for proper functioning. For maintaining synchronization between NoC components we have used handshaking principle between FSMs in our FSM based NoC model.

VIII. CONCLUSION

This work presents formal modeling of NoC using FSM by considering NoC components in detail. Synchronization between different NoC router components is maintained for error-free functioning of a complete NoC system. We have verified the correctness of the model by verifying progress property and synchronization between NoC components. The transfer of packets between routers is also verified. Verification of starvation-freedom considering fixed-priority policy and round-robin policy are presented. Due to the state space explosion problem, verifying globally dependent properties like deadlock is practically not feasible using an intricate NoC model with the help of a model checker. The properties internal to a router are verified considering the individual router. Active windows are introduced to verify properties that involve communication between two routers as encoding numbers of NoC routers with detailed model results in the state space explosion. As the formal verification process is very time expensive, we have used thread level parallelism in our experiments to verify routers individually and active windows corresponding to each router in an NoC. Experimental results significantly improve verification time considering thread level parallelism over its equivalent serial execution.

REFERENCES

[1] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "A generic model for formally verifying NoC communication architectures: A case study," in *Proc. NOCS*, May 2007, pp. 127–136.

[2] D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz, "Executable formal specification and validation of NoC communication infrastructures," in *Proc. 21st Annu. Symp. Integr. Circuits Syst. Design*, 2008, pp. 176–181.

[3] S. Ray and R. K. Brayton, "Scalable progress verification in credit-based flow-control systems," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 905–910.

[4] D. E. Holcomb, A. Gotmanov, M. Kishinevsky, and S. A. Seshia, "Compositional performance verification of NoC designs," in *Proc. MEMCODE*, Jul. 2012, pp. 1–10.

[5] S. Das, C. Karfa, and S. Biswas, "xMAS based accurate modeling and progress verification of NoCs," in *Proc. 21st Int. Symp. VLSI Design Test (VDAT)*, Jun. 2017, pp. 792–804.

[6] V. A. Palaniveloo and A. Sowmya, "Application of formal methods for system-level verification of network on chip," in *Proc. ISVLSI*, Jul. 2011, pp. 162–169.

[7] Y. Chen, W. Su, P. Hsiung, Y. Lan, Y. Hu, and S. Chen, "Formal modeling and verification for network-on-chip," in *Proc. Int. Conf. Green Circuits Syst.*, Jun. 2010, pp. 299–304.

[8] S. El-Ashry, M. Khamis, H. Ibrahim, A. Shalaby, M. Abdelsalam, and M. W. El-Kharashi, "On error injection for NoC platforms: A UVM-based generic verification environment," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1137–1150, May 2020.

[9] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Proc. Int. Conf. Comput. Aided Verification*. London, U.K.: Springer, 1999, pp. 495–499.

[10] S. Das and C. Karfa, *Formal Modeling and Verification of Starvation-Freedom in NoCs*. Singapore: Springer, Jan. 2022, pp. 101–114.

[11] C. Baier and J. Katoen, *Principles of Model Checking* (Representation and Mind Series). Cambridge, MA, USA: MIT Press, 2008.

[12] S.-F. Kuo and C.-W. Wu, "Symbiotic controller design using a memory-based FSM model," in *Proc. IEEE 27th Int. Symp. Ind. Electron. (ISIE)*, Jun. 2018, pp. 874–879.

[13] X. Wan and G. Liu, "Complexity of constructing FSM model of artifact lifecycle," in *Proc. 10th Int. Conf. Semantics, Knowl. Grids*, Aug. 2014, pp. 29–32.

[14] M. Miroschnyk, A. Shkil, D. Rakhlis, E. Kulak, I. Filippenko, and M. Malakhov, "Hardware implementation of timed logical control FSM," in *Proc. IEEE East-West Design Test Symp. (EWDTS)*, Sep. 2020, pp. 1–6.

[15] I. Zuzak, I. Budiselic, and G. Delac, "Formal modeling of restful systems using finite-state machines," in *Web Engineering* (Lecture Notes in Computer Science), S. Auer, O. Díaz, and G. A. Papadopoulos, Eds. Berlin, Germany: Springer, 2011, pp. 346–360.

[16] T. He and H. Miao, "Modeling and composition of web application components using extended FSM," in *Proc. 4th Int. Conf. Natural Comput.*, 2008, pp. 363–368.

[17] S. Lee, S. Yoo, and K. Choi, "An intra-task dynamic voltage scaling method for SoC design with hierarchical FSM and synchronous dataflow model," in *Proc. Int. Symp. Low Power Electron. Design*, 2002, pp. 84–87.

[18] H. Mohanty, J. Mulchandani, D. Chenthati, and R. K. Shyamasundar, "Modeling web services with FSM modules," in *Proc. 1st Asia Int. Conf. Model. Simul. (AMS)*, Mar. 2007, pp. 100–105.

[19] D. Kim and S. Ha, "Asynchronous interaction between FSM and dataflow models," in *Proc. 6th Int. Conf. VLSI CAD*, 1999, pp. 103–106.

[20] P. Linz, *An Introduction to Formal Language and Automata*, 5th ed. Burlington, MA, USA: Jones Bartlett Learning, 2012.

[21] S. Das, C. Karfa, and S. Biswas, "Formal modeling of network-on-chip using CFSM and its application in detecting deadlock," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 1016–1029, Apr. 2020.

[22] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, 1st ed. Burlington, MA, USA: Morgan Kaufmann, 2003.

[23] G. J. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, May 1997.

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

[25] S. N. Huda, "Semantic on PROMELA dynamic process creation in concurrent systems," in *Proc. Int. Conf. Instrum., Control Autom. (ICA)*, Aug. 2016, pp. 44–47.

[26] Y.-C. Lan, S.-H. Lo, Y.-C. Lin, Y.-H. Hu, and S.-J. Chen, "BiNoC: A bidirectional NoC architecture with dynamic self-reconfigurable channel," in *Proc. 3rd ACM/IEEE Int. Symp. Netw.-Chip*, May 2009, pp. 266–275.

[27] P.-A. Hsiung and F. Wang, "A state graph manipulator tool for real-time system specification and verification," in *Proc. 5th Int. Conf. Real-Time Comput. Syst. Appl.*, 1998, pp. 181–188.

[28] Z. Zhang, "Verification methodologies for fault-tolerant network-on-chip systems," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Utah, Salt Lake, UT, USA, 2016.

[29] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras, "xMAS: Quick formal modeling of communication fabrics to enable verification," *IEEE Des. Test Comput.*, vol. 29, no. 3, pp. 80–88, Jun. 2012.

[30] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras, "Quick formal modeling of communication fabrics to enable verification," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Jun. 2010, pp. 42–49.

[31] A. Gotmanov, S. Chatterjee, and M. Kishinevsky, "Verifying deadlock-freedom of communication fabrics," in *Proc. 12th Int. Conf. Verification, Model Checking, Abstract Interpretation*. Berlin, Germany: Springer, 2011, pp. 214–231.

[32] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh, "ADVOCAT: Automated deadlock verification for on-chip cache coherence and interconnects," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 1640–1645.

[33] S. J. C. Joosten and J. Schmaltz, "Automatic extraction of micro-architectural models of communication fabrics from register transfer level designs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1413–1418.

[34] S. J. C. Joosten and J. Schmaltz, "Scalable liveness verification for communication fabrics," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[35] F. Burns, D. Sokolov, and A. Yakovlev, "GALS synthesis and verification for xMAS models," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1419–1424.

[36] F. Verbeek and J. Schmaltz, "Towards the formal verification of cache coherency at the architectural level," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 3, pp. 1–16, Jul. 2012.

[37] P. van Wesel and J. Schmaltz, "Formal micro-architectural analysis of on-chip ring networks," in *Proc. 55th Annu. Design Autom. Conf.*, Jun. 2018, p. 94.

[38] G. Di Guglielmo, F. Fummi, G. Pravadelli, S. Soffia, and M. Roveri, "Semi-formal functional verification by EFSM traversing via NuSMV," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Jun. 2010, pp. 58–65.

SURAJIT DAS received the B.E. degree in computer science and engineering from Jorhat Engineering College (JEC), Assam, India, the M.Tech. degree in computer engineering from the Malaviya National Institute of Technology (MNIT), Rajasthan, India, and the Ph.D. degree from the IIT Guwahati, Assam. He has four years of experience in the software industry. Currently, he is an Assistant Professor with the Department of Computer Science and Engineering, GITAM School of Technology, Bengaluru, India. He has developed a formal simulator for the detection of application specific deadlock with reporting of deadlock scenarios in NoC. He has published his research works in reputed journals. His research interests include computer architecture, formal verification, and machine learning based techniques.

CHANDAN KARFA (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science and engineering from IIT Kharagpur. He was a Senior Research and Development Engineer at Synopsys (India) Pvt. Ltd., for five years. He is currently an Associate Professor with the Department of Computer Science and Engineering, IIT Guwahati. He has published more than 50 research papers in reputed international journals and conferences. His research interests include formal verification, high-level synthesis, hardware security, and formal methods. He has received the Qualcomm Faculty Award from Qualcomm in 2021, the Technoinventor Award by the India Electronics and Semiconductor Association in 2014, the Innovative Student Projects Award from the Indian National Academy of Engineers in 2008 and 2013, the Best Paper Awards in ADCOM Conference in 2007 and in I-CARE conference in 2013, and the Microsoft Research India Ph.D. Fellowship in 2008.

SANTOSH BISWAS (Senior Member, IEEE) received the B.E. degree in computer science and engineering from NIT Durgapur, in 2001, and the M.S. degree in electrical engineering and the Ph.D. degree in computer science and engineering from IIT Kharagpur, in 2004 and 2008, respectively. He is currently a Professor and the Head of The Department with the Department of Electrical Engineering and Computer Science, IIT Bhilai. He has been involved in several research projects sponsored by industry and Government agencies. He is engaged with academic as well as industry sponsored research related to VLSI testing and design for testability. He has published about 170 research articles. His research interests include VLSI testing, fault tolerance, network security, discrete-event systems, and embedded systems.