Simulation Modelling Practice and Theory 77 (2017) 272–291

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Action-Level Real-Time Network-on-Chip Modeling

Soroosh Gholami, Hessam S. Sarjoughian*

School of Computing, Informatics, and Decision Systems Engineering, Arizona State University Tempe, Arizona, United States

A R T I C L E I N F O

Article history:
Received 31 January 2017
Revised 28 June 2017
Accepted 30 June 2017

Keywords:
Action-level modeling
DEVS
Network-on-Chip
Real-time

A B S T R A C T

To simulate time-constrained operations and scheduling for Network-on-Chip (NoC) systems, we introduce a new set of component specifications at flit level grounded in Action-Level Real-Time DEVS formalism. These models capture the dynamics of NoC systems through action-based behavior under strict execution time intervals. These DEVS-based models are well-suited for development and simulation of asynchronous NoC architectures. This is achieved by extending the DEVS-Suite simulator to support real-time executions of ALRT-DEVS models. Representative simulation models capturing structure and behavior of prototypical Mesh NoC systems are developed. A set of experiments are designed, implemented, executed, and analyzed to show the kind of real-time simulation capabilities that can be achieved for Network-on-Chip systems.

1. Introduction

A Network-on-Chip (NoC) is a communication subsystem which connects components of a System-on-Chip (SoC) such as processing elements, memory blocks, and programmable I/O together [1]. Data stream communications between SoC elements are packetized and communicated through the network as flits. Flits from sources are routed in the network using NoC (oblivious or adaptive) routing algorithms to reach their target destinations [2]. Flit data streams are assembled to their original forms at the target nodes.

In [3], authors have proposed four categories for research problems in NoC design: 1) application modeling & optimization, 2) NoC communication architecture analysis, 3) NoC communication architecture evaluation, and 4) NoC design validation and synthesis. Although NoC models developed in this paper have potential to contribute to these categories, our focus here is on the communication architecture evaluation and design validation. NoC modeling and simulation (M&S) is one approach toward design specification and validation. Designers utilize various modeling and simulation tools/methodologies throughout the NoC design process. Each of these tools/methodologies has its place in the NoC design process based on the levels of abstraction it supports. Also, in [4], authors discuss the challenges of peta-scale system design in the future and provide directions for simulating such systems in the future. Among the challenging areas they consider, we focus on performance (with real-time simulation) and scalability (with formal modeling).

In [5], the authors propose four levels of abstraction for NoC. These levels represent the amount of detail the model (and ultimately the simulation) supports. We also refer to each of these abstraction levels as a *stage*, as designers need to go through each of them (in order) to develop a new NoC system from scratch. In the least detailed level (first stage), *interface*, behavior for simple packet delivery and functionality of network interfaces are accounted for. This kind of simulation is used

* Corresponding author.
E-mail addresses: sgholami@asu.edu (S. Gholami), sarjoughian@asu.edu (H.S. Sarjoughian).
URL: https://acims.asu.edu/ (H.S. Sarjoughian)

http://dx.doi.org/10.1016/j.simpat.2017.06.004
1569-190X/© 2017 Elsevier B.V. All rights reserved.

in order to evaluate general dynamics of NoCs. In the second stage, *capacity*, resource constraints (e.g., bandwidth, injection rates, and buffering) are added to support performance evaluation. In the third stage, *flit*, all components related to the flit abstraction such as arbiters, buffers, and switches are modeled individually. Flit level simulation can provide accurate latency analysis based on individual flit deliveries. The results of this type is expressed in cycle times. In the last (fourth) stage, *hardware*, absolute timing and design considerations such as physical area or implementation costs are accounted for. Absolute timing is measured in the form of fractions of seconds, which provide greater insight into the operation of the hardware.

Some NoC modeling frameworks are grounded in formalism-based modeling upon which the code-based modeling (representation of that model in programming languages) is achieved. Considering the overhead of model-based engineering, the "what can be the benefit of formalism-based modeling?" question inevitably is asked. One reason is the flexibility and modularity provided by formalism-based modeling. Unlike code-based models, modeling formalisms allow models to be systematically extended and changed. This is useful since the model may undergo non-trivial design changes in early analysis and design stages (based on simulation results) which benefits from extensibility and modifiability; for example in embedded systems, designing controllers depends on the system architecture and also the components with which the controller interacts. The other benefit is that formalism-based modeling establishes a formal process for describing the system and constructing the models. Models are less error-prone (although formal verification is needed if one needs formal proof) and generic (as opposed to platform-specific programming languages). Few formalisms (e.g. Dart [6]) provide the additional capability needed for models to be simulated in real-time and replaced with actual component/subsystems operating within a larger system or environment. This provides us with the capability to test the (sub)system under physical operational settings. Using real-time simulation, one can test a subsystem under scenarios which are too costly or impossible to replicate in a logical-time simulation. Such real-time M&S contributes in making richer decisions at earlier stages of system design process.

This paper proposes a new approach for modeling and simulating Network-on-Chip systems using real-time, hierarchical specifications where behavior of individual components are defined in terms of a set of actions each with their own timing constraints. NoC is specified using ALRT-DEVS. We brought forth the theory of ALRT-DEVS in [7], which enables modeling of systems using actions and time windows (details are provided in Section 2). This abstract specification technique is concretized for NoC to model the dynamics of this system in a time sensitive manner. We have developed an extension to the DEVS-Suite simulator [8,9] to enable implementation and execution of actions under worst case execution constraints. Leveraging the real-time modeling capability with real-time execution, a simulated NoC can mirror its counterpart subject to computational efficiency of host simulation platform. Models developed using ALRT-DEVS, because of their capability in capturing the dynamic behavior of the system with respect to time, are well-suited to represent asynchronous logic. A significance of this capability in NoC platform is support for GALS (Globally Asynchronous Locally Synchronous) architecture.

For the NoC models presented here, we focus on flit-level abstraction described earlier. These models define both structure (such as the topology, virtual channels, buffers, crossbars, and links) and behavior (such as the switching policy, routing algorithm, and flow control mechanism) of NoC as a set of composable model components. NoC model component specifications collectively should allow defining behavior of model components in terms of time-constrained actions. Also, these models should lend themselves to be simulated in real-time. These NoC components should have sound behavioral specifications – i.e., actions executed subject to Worst Case Execution Time (WCET) in real-time.

In the M&S framework introduced in this paper, the simulation protocol is independent of the modeling formalism. In many simulation tools the model with its execution protocol are tightly coupled. The NoC simulation models formalized, developed, and validated in the rest of this paper are independent of the simulation protocol used for executing it. We extended the DEVS-Suite simulator to support both modeling and simulation of NoCs. These specifications can be executed in real-time using the extension built for DEVS-Suite if the executing platform is capable of handling the amount of computation needed. The NoC models developed in Section 3 are implemented and tested using scenarios that encompass timeliness and accuracy. Timeliness of execution refers to whether the simulation engine is capable of executing the model within the real-time bounds defined for it. The accuracy of the models are measured by comparing performance measures received from the simulator with the expected results. We implemented all the NoC component models in the extended DEVS-Suite simulator [8] and validated them with experiments.

The actions belonging to the NoC models can be specified such that their executions are known to meet their deadlines as measured in a real-time simulator. Any scenario in which actions are not completed within their Worse Case Execution Time deadlines show either a system failure or a degraded quality of service. Whether or not a model can be executed fully in real-time is decided by the computational demands of the model (within a period of time) and the computational power of the executing platform. At any point during the simulation, if the latter is smaller than the former, deadline misses are expected and timeliness is not satisfied.

The capabilities of the ALRT-DEVS modeling approach is used to create a novel flit-level NoC model. All elements of this model are developed and simulated in the DEVS-Suite simulator. The approach along with data reporting/analysis is then compared with BookSim (v. 2.0) [10], a prototypical tool dedicated to NoC simulation. We show how NoC models DEVS-Suite function similar to well-known simulators. Our intention is to show while ALRT-DEVS is more demanding compared with BookSim, its formalism leads to developing rigorous designs. Multiresolution modeling and model checking are among the benefits of an explicit DEVS-based formalis [11,12]. In addition, data reporting/analysis is far more flexible than that of code-based environments such as BookSim.

The rest of the paper is organized as follows. Section 2 briefly describes the concepts and approaches used for modeling Network-on-Chip systems. In Section 3, a generic flit-level component-by-component model of NoC is designed and used for developing example models. In Section 4, some experiments are devised. Using these experiments, we compare NoC real-time modeling and simulation with a representative of well-known NoC simulation tools. In Section 5 we mention some of the basic and state of the art methods and tools for modeling and simulation of NoC. Finally, a summary and future work is described in Section 6.

2. Background

2.1. Modeling actions with time constraints using ALRT-DEVS

We introduced modeling and simulation of Network-on-Chip using Discrete-Event System Specification (DEVS) modeling approach and Real-time Statecharts in [13]. This approach is based on defining behavior of NoC components as actions that must be completed within a time window. Actions with time interval abstractions are defined in RT-DEVS [14]. RT-DEVS brings the concepts of action set (A) for activities that are constrained by time intervals and the time interval function (ti) that maps a time window to each action [15]. These concepts are introduced to DEVS specification in order to bring timing information to DEVS models. Action execution is considered valid if it can be completed within some specified time window by the time interval function. Time interval function defines a range $[x, y]$ within which the execution of action should be completed ($ti : A \rightarrow [x, y], where\ (x, y) \in \mathbb{R}^+_{0,\infty} \times \mathbb{R}^+_{0,\infty} \wedge x < y$). Real-time Statecharts define location and transition concepts [16]. Statecharts can be used to prioritize, change, or even remove actions based on the time remaining to the deadline. The modeler (using domain knowledge) defines scheduling rules in Statechart models; therefore, scheduling decisions are independent of the executing platform.

We introduced the theory of ALRT-DEVS in [7] based on our earlier work on NoC simulation with RT-DEVS and Real-time Statecharts. The formal definition of atomic models in ALRT-DEVS is as follows:

$$ALRT\text{-}DEVS = \langle X, Y, S, A, \Gamma, \Omega, \psi, \lambda, ti \rangle.$$

The input, output, and state sets (X, Y, and S) are the same as those in the Parallel DEVS specification [17]. The action set (A) and time interval function (ti) are defined above. To describe the rest of the sets and functions in this specification, we first need to introduce the concept of location. A location is specified using two primary state variables of DEVS models called *phase* and sigma (σ). Similar to the concept of states in Real-time Statecharts, it contains invariants and entry/exit/do actions. ALRT-DEVS extends the concept of DEVS states with Statachart locations which brings additional useful concepts such as invariants, actions, and time windows (constraints). The functions in ALRT-DEVS specification operate based on location (instead of state). The functions Γ and Ω encapsulate the external and internal transition functions respectively. The activity mapping function (ψ) maps actions to location. Finally, the output function (λ) maps actions to output values. We encourage the reader to refer to ALRT-DEVS specification for details on model syntax and operational semantics [7].

Behavior modeling of NoC is defined using ALRT-DEVS's actions, locations, and time windows for real-time modeling and simulation. ALRT-DEVS introduces locations and time-constrained actions that belong to external and internal transition functions; which ties the concepts of locations, actions, and external/internal transition functions in one specification. Each action is handled independently from the other actions mapped to the same location. Using the concept of locations and invariants, capabilities such as time constraints on states and action priorities are introduced to ALRT-DEVS.

2.2. Real-time simulation

Models with time-constrained actions can be simulated in either logical-time or real-time. The difference between these two simulation protocols has its roots in how the simulation clock is synchronized using a real-time clock (wall clock). The simulation clock is an "abstraction of real-time clock" which itself represents the actual (physical) time it takes for the model to be executed [7,18]. When simulating a real system there is always a correspondence between simulation time and physical time (in both logical-time and real-time simulations). However, in real-time simulation, the pace of simulation and wall clocks are assumed to be equal. This way, time is a representation of physical time as perceived by human operators, which enables simulation replacement in place of the system in an operational environment. The abstract and operational semantics defined by ALRT-DEVS models possess appropriate timing information to be simulated in real-time. However, a real-time simulation engine is still necessary to execute these models in real-time. We developed a simulator protocol in [7]. We introduced and implemented this new simulator protocol in the DEVS-Suite simulator [8]. Both real-time and logical-time simulations can be supported with this simulator. Logical-time is useful for validating NoC models while real-time is used for validating real-time NoC dynamics if and when such simulators are co-executed with physical components. We show the applicability of such scenario (hardware-simulator co-execution) in [19]. The simulation protocol must handle all actions missing their deadline due to simulator's own performance limitation. The compensation activity is dependent on the target system being modeled. For NoC, the compensation may be to discard or move the flits to the end of the queue for reprocessing (depending on the time criticality of the flits). Inadequate simulator's performance will introduce artificial timing inaccuracies which in turn invalidate timing deadlines defined for actions. As mentioned earlier, the flit level and hardware-level NoC models (most detailed) may not be simulatable in real-time due to high computational

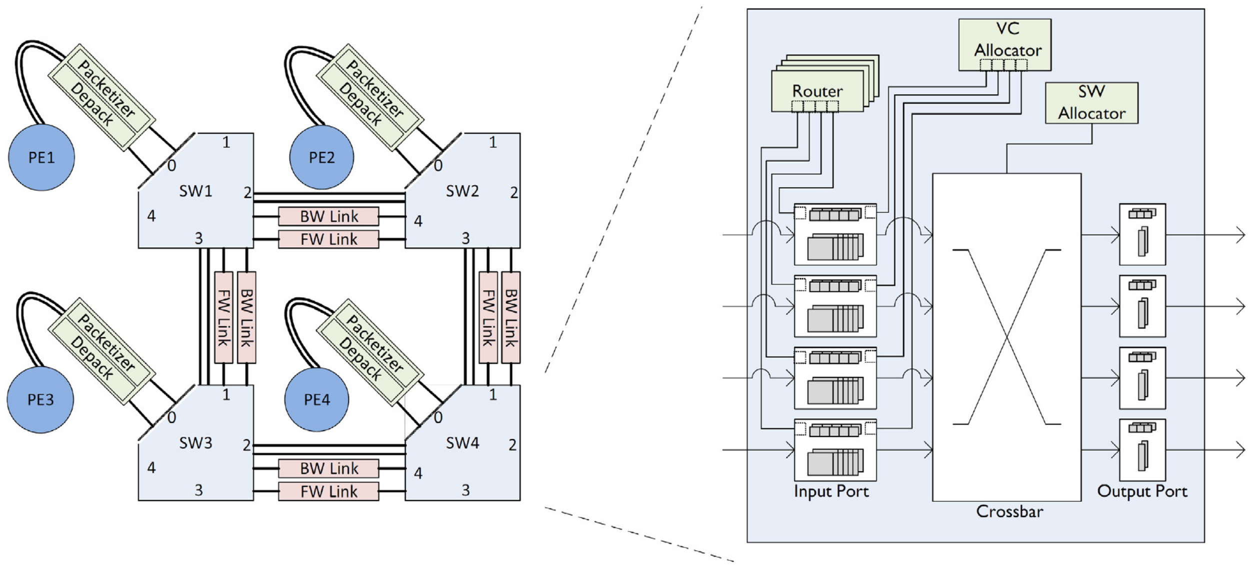


Fig. 1. A schematic of a 2 × 2 mesh NoC structure with its components and connections.

demands. Therefore, the real-time execution is a more appropriate option for more abstract NoC models. These models can be connected and co-simulated with existing systems (simulation, hardware, or software) for requirements verification.

2.3. Network-on-Chip modeling

Fig. 1-left illustrates basic components and connections for a 2× 2 prototypical NoC with a mesh topology. The switch component is shown in greater detail on the right hand side of Fig. 1. The switch-based model of communication has several advantages over regular bus-based approaches. First, packets are routed in the network simultaneously and in most common methods the path is allocated piecewise (link or virtual channel) to the packet as opposed to utilizing a bus-based approach in which the entire bus is allocated and charged for a single message transfer causing less bandwidth and higher power consumption. Second, regular structure of Network-on-Chips makes it easier for hardware engineers to design and implement.

2.4. Asynchronous hardware and NoC modeling

An asynchronous circuit is a sequential logic which is not timed by a clock signal [20]. This form of design gave birth to the world's first asynchronous microprocessor [21] and bendable processor [22]. Asynchronous design does not bound hardware modules to adhere to a clock signal. The progress of operations through the circuit is handled by handshaking and task completion logics. This leads to several advantages. First, there is no requirement for clock signal propagation through large chips. Not being dependent to a clock signal also provides higher performance for function units. Also, parts of the circuit that are not active with any operation will not use energy which lowers the power consumption of the chip. These circuits are more resistance when it comes to electromagnetic interferences and can have elastic pipelines [23]. Of course this approach has some limitations: space-inefficiency, testing/verification difficulties, scarcity of research and development tools, and requirements for higher-valued logic. Asynchronous hardware has entered the field of network-on-chip as well [24]. Globally Asynchronous Locally Synchronous (GALS) is a paradigm in which components of the chip are grouped together in different zones synchronously but there are no synchronization at the global level. This way, NoC can enjoy the benefits of both synchronous and asynchronous logic design paradigms.

Since GALS is a new trend in NoC design, the modeling and simulation frameworks must have support for asynchronous design. What we offer in this paper is a combined modeling and simulation approach which has the basics for this class of a/synchronous NoCs. The unpredictability and self-timeliness of asynchronous logics are well supported.

2.5. Network-on-Chip simulation

With increasing use of Network-on-Chip for embedded and even general purpose software-based systems, the importance of theory-based Modeling and Simulation (M&S) approaches for measuring performance, area of the chip, power consumption, and etc. becomes more apparent. Some of these approaches incorporate integrated M&S in which the model and the simulator are both embedded in a programming language. This approach usually leads to a rigid structure for the simulator (model is not separated from the simulator). Most important academic and commercial NoC simulators use this approach

Table 1
Location description for processing element statechart diagram (Fig. 2).

Location	Phase	do() action; worst case execution time	Entry/exit actions
Process Data	Active	processData(); $w = T_{Max}^{processData}$	–/–
Generate Data	Active	generateData(); $w = T_{Max}^{generateData}$	–/addDataToQ()
Idle	Idle	–	–/–

Table 2
Variables for packetizer.

Variable	Type	Description
outFlitQ	Queue of flits	Packetized flits, ready to be sent out
inData	Data stream	Data stream received from PE to be packetized
outQStatus	Boolean	Holds the status of neighboring switch's input queue

Table 3
Locations for Packetizer's Statechart (Fig. 3).

Location	Phase	do() action; worst case execution time	Entry/exit actions
Packetize	Active	packetizeData(); $w = T_{Max}^{packetizeData}$	–/addToOutputQ()
Waiting	Active	–	–/–
Transmission	Active	sendFlitToSwitch(); $w = T_{Max}^{sendFlitToSwitch}$	–/–
Idle	Idle	–	–/–

to gain efficiency. The other approach is separating the model from the simulator engine which supports wider range of systems but usually are less efficient. A key advantage of this latter approach is principled model specification with loose dependency on simulator implementation.

As mentioned earlier, NoC can be modeled and simulated in four levels: 1) Interface Level, 2) Capacity Level, 3) Flit Level, and 4) Hardware Level. Models with more details (higher resolution) grant more accurate results but require longer execution time. The level of simulation which is used for system analysis is dependent to the stage of design and implementation. In very early stages such as concept-based design, high-level (low resolution) decisions are useful. However, in later stages, concrete design decisions require simulation models at flit level or hardware level.

3. NoC model components

In this section, we show our design of a suite of NoC models. The implementation and evaluation of models are discussed in Section 4. Four major components of SoC are 1) Switch, 2) Network Interface, 3) Bus (contains a set of links and signal lines), and 4) Processing Element. Since software cannot be ignored in simulating the hardware, we include processing element and network interface components in the NoC model as well. In this work, we consider a simple model of Processing Element which acts as a traffic generator and put more emphasis on the networking aspects of the system. Except for the Processing Element, the other three major components are coupled models with several subcomponents at flit level. At this level, we do not account for hardware details such as handshaking schemes and synchronous/asynchronous communication although this can be supported using DEVS as mentioned in Section 5.

It should be noted that this work deals with the simulation of the network aspects of NoC. Therefore, concepts such as workload characterization (as introduced in Section 5) and memory/processor architecture modeling are outside the scope of this NoC model. This simulation can be combined with other DEVS-based simulations on processor architectures (works similar to [25]) and synthetic workloads to include additional lower-level details. We discuss future extensions of this work in Section 6.

In the rest of this section, we describe each component and then detail its model in ALRT-DEVS. Each model has a real-time Statecharts and DEVS-based specifications. Additional details such as descriptions of state variables and valid values are provided in Tables 2 and 4. for the more complex models. These along with their implementations [26] allow designers and users to gain a thorough understanding of how the structures and behaviors of the NoC model components work. The following section contains selected component specifications to show how we can model prototypical NoCs. All the model component specifications we have developed for NoC can be found in [26]. Next, we begin with the simple processing element model and introduce more complex ones step-by-step. The complexity of a model is determined by many factors such as the number of locations, number of state variables, guard conditions, etc.

3.1. Processing element

Processing Element is responsible for processing data. The idea of NoC is to connect these independent cores to each other enabling them to complete a task cooperatively. As mentioned at the beginning of this section, the purpose of this

Table 4
Variable description for input port.

Variable	Type	Description
N_{VCs}	Integer	Number of virtual channels in the network
N_{Ports}	Integer	Number of ports for each switch
vcStatus	a set containing the status of virtual channels. It stores information such as: *output port* (R), *destination VC* (O), *stage* (G), and *pointers* (P)	Holds the status of each virtual channel in the input port. *Output Port* and destination VC hold the destination port and VC, respectively. Stage specifies the current stage of the head flit (*R, VA, SA,* and *ST*). The current length of the queue is held in *size* variable
statusInconsistent	boolean variable	*True* if the internal status of an input port is inconsistent with the value on *statusPorts*
inputQs	a set of input queues for incoming packets (there are N_{VCs} queues)	Array of input queues for each virtual channel
round	integer	Specifies the turn for switch allocation among VCs

$$
\begin{aligned}
&PE = \langle X, Y, S, A, \Gamma, \Omega, \psi, \lambda, ti\rangle\\
&S = \overbrace{\{\texttt{Active, Idle}\}}^{\texttt{phase}} \times \overbrace{\sigma}^{\texttt{sigma}} \times \overbrace{\{0,1\}^*}^{\texttt{incomingDataBuffer}} \times \overbrace{\{0,1\}^*}^{\texttt{outgoingDataBuffer}}\\
&X = \{(\mathtt{fromNI}, \{0,1\}^*), (\mathtt{start}, 1), (\mathtt{stop}, 1)\}\\
&Y = \{(\mathtt{toNI}, \{0,1\}^*)\}\\
&A = \{processData, generateData\}\\
&L = \{L_{ProcessData}, L_{GenerateData}, L_{Idle}\}\\
&\Gamma = \{\delta_{ext,1}, \delta_{ext,2}, \delta_{ext,3}\}\\
&\delta_{ext,1}(location, e, (stop, 1)) = L_{Idle}\\
&\delta_{ext,2}(location, e, (fromNI, \{0,1\}^*)) = L_{ProcessData}\\
&\delta_{ext,3}(location, e, (start, 1)) = L_{GenerateData}\\
&\Omega = \{\delta_{int,1}, \delta_{int,2}\}\\
&\delta_{int,1}(L_{ProcessData}) = L_{GenerateData}\\
&\delta_{int,2}(L_{GenerateData}) = L_{GenerateData}\\
&\lambda(generateData) = (toNI, \{0,1\}^*)\\
&\psi(L_{ProcessData}) = \{processData\}\\
&\psi(L_{GenerateData}) = \{generateData\}\\
&\psi(L_{Idle}) = \{\}\\
&ti_{processData} = [T^{processData}_{Min}, T^{processData}_{Max}]\\
&ti_{generateData} = [T^{generateData}_{Min}, T^{generateData}_{Max}]
\end{aligned}
$$

Listing 1. Model of the processing element in ALRT-DEVS.

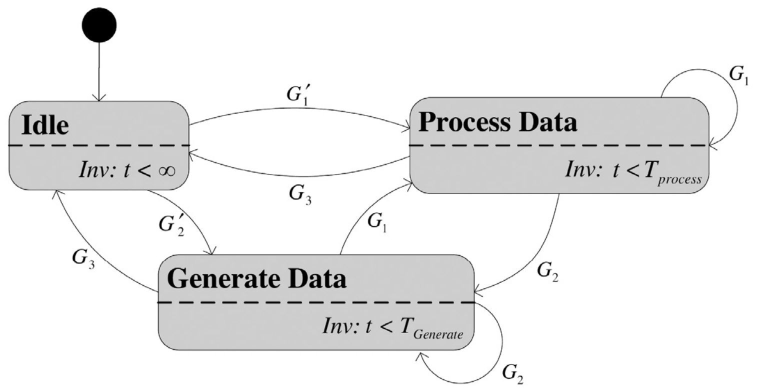
work is on the networking aspects of NoC; therefore, the PE model contains simplified internal structure and behavior with accurate I/O and timing information. Processing element models can differ from one another which can incorporate heterogeneity in NoC M&S. Listing 1 provides the formal model of PE in DEVS.

In the specification presented for PE (Listing 1), the *start* and *stop* input ports, are placed in the model to start or stop the simulation. They only receive one type of value (1) while for *fromNI* and *toNI* ports we used regular expression notation. Therefore, an expression such as $\{0, 1\}^*$ is a sequence of 0s and 1s which is an abstraction for flits (see Section 3.3).

Fig. 2 presents the Statecharts diagram for the PE model. This view is completed with phase, entry/exit/primary action, and the execution times shown in Table 1 to provide a complete model. Worst case execution time (w) for an action is equal to the upper-bound of its execution time interval. Processing elements generate and process data through *generateData* (in **Generate Data** location) and *processData* actions (in **Process Data** location), respectively. This component goes to *Idle* state if no data is waiting to be sent out or be processed. The concept of *Infinity* in ALRT-DEVS models (in the PE model as an invariant for the Idle location), means infinite waiting for interrupt/external events. A model with a time interval of *Infinity* waits indefinitely for an external event before being allowed to change its location/state.

3.2. Network interface

This component is the interface between PE and Switch. It is responsible for packetizing data streams (into flits) and depacketizing flits (into data streams). In a real-time model of NoC, these two actions should be time-bounded as well. Variable length data is an obstacle for predictability and consequently for real-time modeling given arbitrary hardware and software realizations. We modeled Network interface as a coupled model (see Fig. 1). The inner components are: packetizer and depacketizer. The packetizer component is specified below.



$$G_1 = [incomingDataBuffer\ is\ nonempty] \qquad G_1' = Event_{External}[G_1]$$
$$G_2 = [outgoingDataBuffer\ is\ nonempty] \qquad G_2' = Event_{External}[G_2]$$
$$G_3 = [incomingDataBuffer\ is\ empty \wedge outgoingDataBuffer\ is\ empty]$$

Fig. 2. Real-time statecharts for processing element.

$$
\begin{aligned}
&S = \overbrace{\{Active, Idle\}}^{\texttt{phase}} \times \overbrace{\sigma}^{\texttt{sigma}} \times \overbrace{\{0,1\}^*}^{\texttt{outFlitQ}} \times \overbrace{\{0,1\}^*}^{\texttt{inData}} \times \overbrace{\{0,1\}}^{\texttt{outQStatus}}\\
&X = \{(fromPE, \{0,1\}^*), (statusFromSW, \{Ok, Nok\})\}\\
&Y = \{(toSW, \{0,1\}^*)\}\\
&A = \{packetizeData, sendFlitToSwitch\}\\
&L = \{L_{Packetize}, L_{Waiting}, L_{Idle}, L_{Transmission}\}\\
&\Gamma = \{\delta_{ext,1}, \delta_{ext,2}\}\\
&\delta_{ext,1}(L_{Idle}, e, (fromPE, \{0,1\}^*)) = L_{Packetize}\\
&\delta_{ext,2}(L_{Waiting}, e, (statusFromSW, Ok)) = L_{Transmission}\\
&\Omega = \{\delta_{int,1}, \delta_{int,2}\}\\
&\delta_{int,1}(L_{Packetize}) = \begin{cases} L_{Waiting} & [if\ outQStatus = 0] \\ L_{Transmission} & [O.W.] \end{cases}\\
&\delta_{int,2}(L_{Transmission}) = \begin{cases} L_{Idle} & [outFlitQ\ and\ inData\ are\ empty] \\ L_{Packetize} & [O.W.] \end{cases}\\
&\lambda(sendFlitToSwitch) = (toSW, outFlitQ.remove())\\
&\psi(L_{Packetize}) = \{packetize\}\\
&\psi(L_{Waiting}) = \{\}\\
&\psi(L_{Transmission}) = \{sendFlitToSwitch\}\\
&\psi(L_{Idle}) = \{\}\\
&ti_{packetizeData} = [T_{Min}^{packetizeData}, T_{Max}^{packetizeData}]\\
&ti_{sendFlitToSwitch} = [T_{Min}^{sendFlitToSwitch}, T_{Max}^{sendFlitToSwitch}]
\end{aligned}
$$

Listing 2. Model of the packetizer (a component of network interface) in ALRT-DEVS.

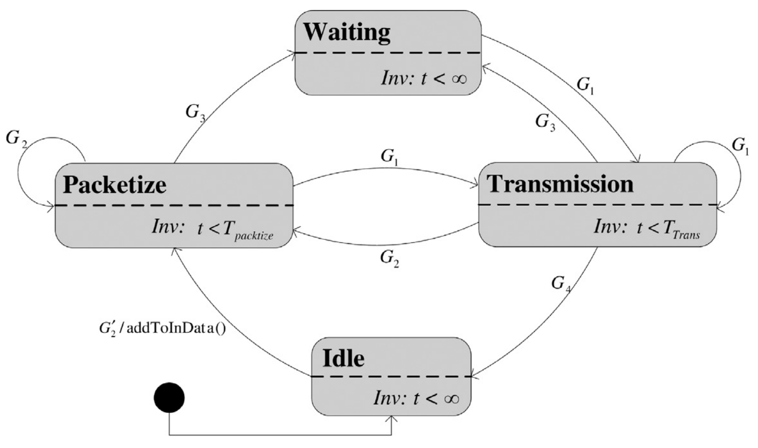
3.2.1. Packetizer

This component packetizes the data streams it receives from the adjacent processing element. It also sets flit header and footer while each flit is created and sent to the neighboring switch. The packetizer component must not overwhelm the switch, therefore, it receives flow control signals from the switch. When switch input queue is full, no new flit is packetized and sent to the switch. The specification for this component is presented in Listing 2.

As specified in Table 2, *outQStatus* holds the status of the switch. This variable is changed if an external event is received from *statusFromSW* port. This status variable, ensures that switch's input queue does not overflow. The Statecharts model of this component in Fig. 3 demonstrates this behavior. Table 3 specifies locations, actions, and their time constraints. In order to prevent overflow from happening, network interface waits infinitely in the *Waiting* location until it receives an *OK* signal from its designated switch. The specification for depacketizer can be found in [26].

3.3. Switch

Switch has the key responsibility of making routing and flow control decisions for the flits it receives from other switches or its designated network interface. This includes deciding about the priority of flits over each other and choosing the best



$$G_1 = [outFlitQ\ is\ nonempty]$$
$$G_2 = [inData\ is\ nonempty] \qquad G_2' = Event_{External}[G_2]$$
$$G_3 = [outFlitQ\ is\ nonempty \wedge outQStatus = \text{“}Nok\text{”}]$$
$$G_4 = [inData\ is\ empty \wedge outFlitQ\ is\ empty]$$

Fig. 3. Real-time statecharts diagram for packetizer.

paths considering the local information (such as network congestion, link capacity, resource availability, and flit deadlines). Simplicity and light-weight communication patterns are the two important properties of routing algorithms that should be considered when choosing one. Aside from time-bounded flits, in-order flit delivery could also be important in some NoCs. A switch has several internal components for input ports, output ports, and a crossbar to connect these input ports to output ones. These internal components are shown in Fig. 1. Flow control mechanism is implemented in this component which is further illustrated in the switch's subcomponents. For the switch model developed here, we chose the on/off flow control mechanism. Input port (as one of the internal components of the switch) is specified below.

3.3.1. Input port

The input port component is responsible for receiving incoming flits from the neighboring switches or NI. Each input port contains status arrays and flit queues equal to the number of the virtual channels of that network. Each status array holds information about the head flit of its corresponding queue. In addition, routing and allocating operations are done via actions in this component instead of embedding them in separate router/allocator components. Flow control mechanism is also implemented in this component. Input queues send *off* signals to the upstream switch whenever the input queue has passed the upper bound. The *on* signal is sent when the number of waiting flits drop below the lower bound of the buffer. Next, the atomic model for the switch's input port is specified in Listing 3.

The definition of state, in addition to phase and sigma, contains input queues, head flit (under service), turn (which virtual channel to service), status of input queues (for flow control purposes), and the *statusInconsistent* boolean variable (as specified in Listing 3). All these variables are described in Table 4. We specified two types of output ports for this component: one for transmitting flits (*outport*) and one for status signals (*statusPort*). The status signals are used for flow control in the network. In this model, we incorporate on/off flow control mechanism in which “OK” or “NOK” signals are sent out based on the current status of the input port. High and low thresholds are predefined for the input buffer. The “OK” signal is transmitted when the number of items is less than the low threshold and the “NOK” is transmitted when the buffer index becomes greater than the high threshold.

The guard conditions are put in square brackets (similar to well-known formalisms such as Timed Automata). All the guards (such as G_R) are specified in Fig. 4. The only external event is the receipt of a new flit which adds it to the *inputQs* (see Table 4). The only internal event results in sending one flit to the crossbar and passing the turn to the next input queue based on round robin scheduling. In case of empty queues the component goes into *idle* phase. Also, in case of inconsistency between internal and external status (*statusInconsistent* changes to true) the new status signal is sent to neighboring switches. The Statecharts diagram for input port is presented in Fig. 4 and Table 5. The table contains phase, actions, and execution time information for each location of the Statecharts. The guard for each transition is specified in the figure. Guards with lower numbers have priority over guards with higher numbers. Therefore, if $guard_i$ and $guard_j$ are both satisfied and $i < j$, the first transition (guarded by i) is taken (at any instance of time only one of the guards G_R, G_{VA}, G_{SW}, and G_{ST} can be satisfied). With this in mind, it is obvious that status transmission has the highest priority among all actions in this model of the Input Port. Thus, whenever *statusInconsistent* is true, the model changes its location to *Status Transmission* to

$$S = \overbrace{\{Active, Idle\}}^{\texttt{phase}} \times \overbrace{\sigma}^{\texttt{sigma}} \times \overbrace{\{0,1\}^{*^{N_{VCs}}}}^{inputQs} \times \overbrace{\{0,1\}^{*}}^{\texttt{outFlit}} \times \overbrace{\mathbb{N}}^{\texttt{round}} \times \overbrace{\{GROPC\}^{N_{VCs}}}^{vcStatus} \times \overbrace{\{true, false\}}^{statusInconsistent}$$

`in which:` $GROPC \in \mathbb{N}^5$`, G (global state), R (route), O (output VC), P (head/tail pointers), C (credits)`

$X = \{(inport[0..N_{VCs}], \{0,1\}^*)\}$

$Y = \{(outport, \{0,1\}^*), (statusPort[0..N_{VCs}], \{Ok, Nok\})\}$

$A = \{statusTransmitter, router, vcAllocation, swAllocation, swTraversal\}$

$L = \{L_{StatusTransmission}, L_{Routing}, L_{VCAllocation}, L_{SwitchAllocation}, L_{SwitchTraversal}, L_{Idle}\}$

$\Gamma = \{\delta_{ext,1}\}$

$$\delta_{ext,1}(location, e, (inport[i], X)) = \begin{cases} L_{Routing} & [if\ location\ is\ Idle] \\ location & [O.W.] \end{cases}$$

$\Omega = \{\delta_{int,1}, \delta_{int,2}, \delta_{int,3}, \delta_{int,4}, \delta_{int,5}, \delta_{int,6}\}$

$\delta_{int,1}(location) = L_{StatusTransmission} \quad [if\ statusInconsistent\ is\ True]$

$$\delta_{int,2}(StatusTransmission) = \begin{cases} L_{Routing} & [G_R] \\ L_{VCAllocation} & [G_{VA}] \\ L_{SwitchAllocation} & [G_{SW}] \\ L_{SwitchTraversal} & [G_{ST}] \end{cases}$$

$\delta_{int,3}(L_{Routing}) = L_{VCAllocation} \quad [G_2]$

$\delta_{int,4}(L_{VCAllocation}) = L_{SwitchAllocation} \quad [\neg(G_1 \wedge G_4)]$

$\delta_{int,5}(L_{SwitchAllocation}) = L_{SwitchTraversal} \quad [\neg(G_1 \wedge G_5)]$

$$\delta_{int,6}(L_{SwitchTraversal}) = \begin{cases} L_{Routing} & [\neg G_6] \\ L_{Idle} & [G_6] \end{cases}$$

$\lambda(statusTransmitter) = (statusPort[i], maxSize - vcStatus.P) \quad [0 \le i < N_{VCs}]$

$\lambda(swTraversal) = (toSW, outFlit)$

$\psi(L_{StatusTransmission}) = \{statusTransmitter\}$

$\psi(L_{Routing}) = \{router\}$

$\psi(L_{VCAllocation}) = \{vcAllocation\}$

$\psi(L_{SwitchAllocation}) = \{swAllocation\}$

$\psi(L_{SwitchTraversal}) = \{swTraversal\}$

$\psi(L_{Idle}) = \{\}$

Listing 3. Model of the input port (a component of switch) in ALRT-DEVS.

Table 5
Location description for input port statechart diagram (Fig. 4).

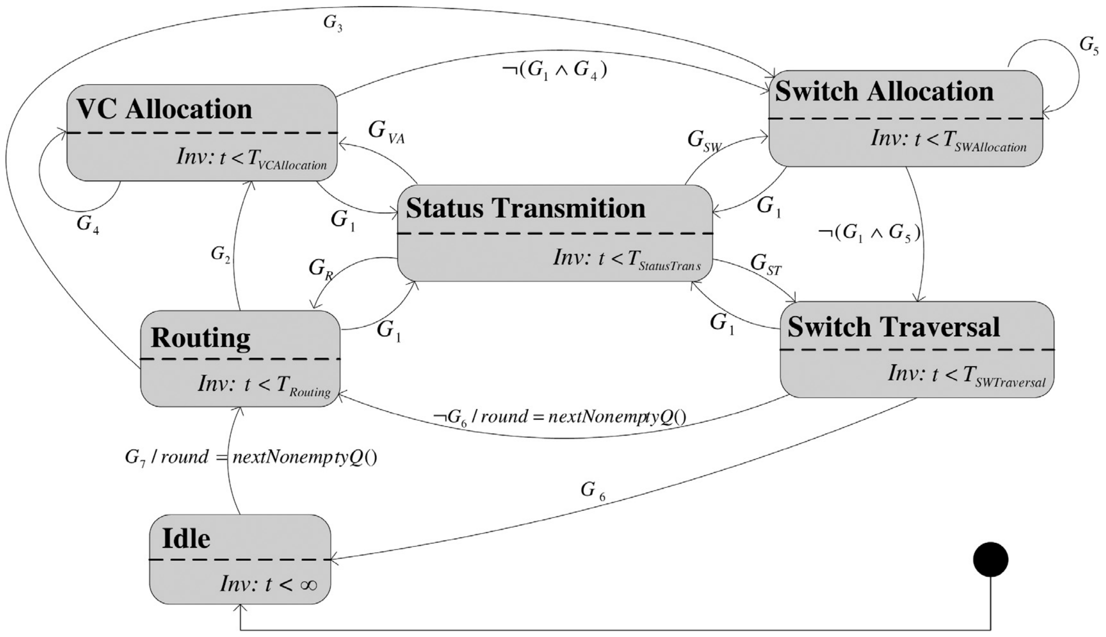
Location	Phase	do() action	Entry/exit actions
Routing	Active	router() $w = T_{Max}^{router}$	updateStatusArray(loc)/ updateStatusArray(outPort)
Switch Allocation	Active	swAllocation() $w = T_{Max}^{swAllocation}$	updateStatusArray()/–
Switch Traversal	Active	swTraversal() $w = T_{Max}^{swTraversal}$	updateStatusArray(loc)/–
VC Allocation	Active	vcAllocation() $w = T_{Max}^{vcAllocation}$	updateStatusArray(loc)/ updateStatusArray(vc)
Idle	Idle	–	–/–
Status Transmission	Active	statusTransmitter() $w = T_{Max}^{statusTransmitter}$	–/–

synchronize internal and external status signals. Routing, allocations, and traversal actions are executed in sequence based on the guards set on transitions in Fig. 4.

With the models for the remaining elements for NoC provided in [26], we have a set of specifications for modeling NoC systems. We developed these models in DEVS-Suite and executed selected NoC configurations. We then compared the results with those obtained with BookSim. These comparisons are meant to showcase the method by which each conduct modeling and simulation of NoC models. Some of these experiments are presented in Section 4.

3.4. Model construction

To construct NoC models, the network interface (*ni0, ni1*), switch (*sw0, sw1*), and bus (*Hbus0*) coupled models and the processing elements (*pe0, pe1*) atomic models are hierarchically connected via their input/output ports. Various methods of connecting these components result in different topologies. For simplicity, a 2-node with mesh topology representing a $n \times m$ NoC is considered (see Fig. 5). Each atomic model of processing element is connected to a coupled model of network



$$G_1 = [statusInconsistent\ is\ true]$$
$$G_2 = [vcStatus.R\ is\ set \wedge N_{VCs} > 1]$$
$$G_3 = [vcStatus.R\ is\ set \wedge N_{VCs} == 1]$$
$$G_4 = [vcStatus.O\ is\ not\ set] \qquad G_R = [vcStatus.G = \text{“}R\text{”}]$$
$$G_5 = [swAllocation\ unsuccessful] \qquad G_{VA} = [vcStatus.G = \text{“}VA\text{”}]$$
$$G_6 = [all\ inputQs\ are\ empty] \qquad G_{SW} = [vcStatus.G = \text{“}SW\text{”}]$$
$$G_7 = Event_{External}[\neg G_6] \qquad G_{ST} = [vcStatus.G = \text{“}ST\text{”}]$$

Fig. 4. Real-time statecharts diagram for input port.

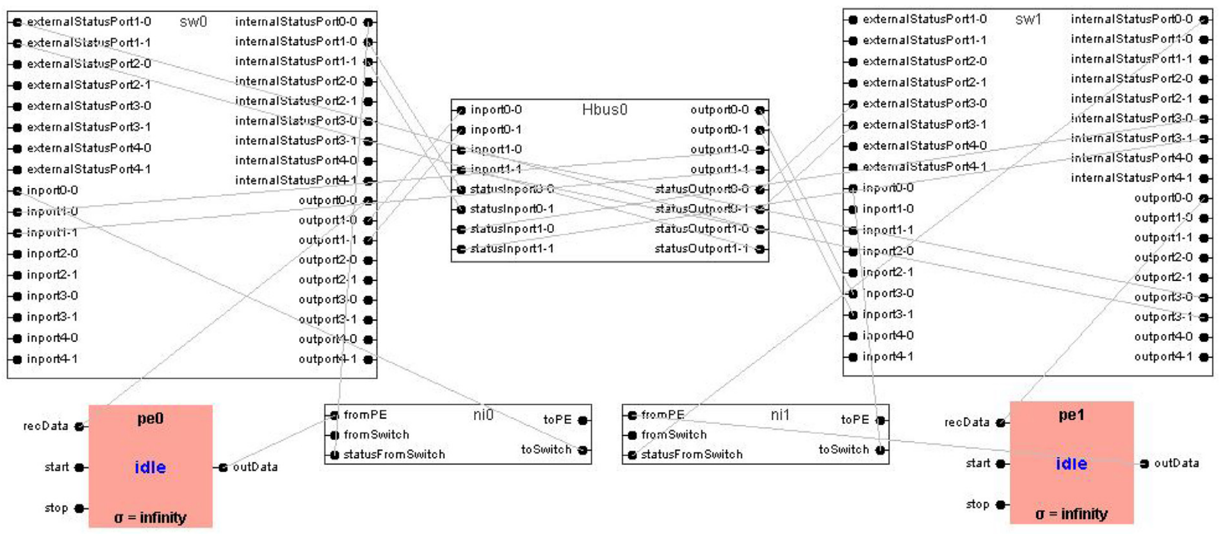


Fig. 5. A partial hierarchical mesh NoC system model.

```
num_vcs    =  2;             //Number of Virtual Channels
vc_buf_size = 8;             //Number of buffers for each channel
wait_for_tail_credit = 1;

vc_allocator = separable_input_first;     //Virtual channel allocation policy
sw_allocator = separable_input_first;     //Switch allocation policy

credit_delay  =  1;     //One cycle for credits to reach the next switch
routing_delay =  1;     //One cycle for routing
vc_alloc_delay = 1;     //One cycle for vc allocation
sw_alloc_delay = 1;     //One cycle for switch allocation
st_final_delay = 2;     //One cycle for switch traversal, one for link traversal
input_speedup =  1;     //Input/output ratio in crossbar
output_speedup = 1;
interval_speedup = 1;

sim_type  =  latency;      //Let all packets drain
warmup_period =  0;
sample_period =  1000;
sim_count =  5;            //Perform each simulation 5 times

topology  =  mesh;
k = 3;                  //Number of nodes in each dimension
n = 2;                  //Number of dimensions

routing_function  =  dor;      //Dimension-order-routing
const_flits_per_packet = 1;    //Flits per packet
use_read_write =  0;
traffic = uniform;
injection_rate =  0.1;         //One flit per 10 cycles
```

Listing 4. Model of the packetizer (a component of network interface) in ALRT-DEVS.

interface which communicates with a switch. PEs and NIs communicate data streams while NI-switch and switch-switch interactions are based on flits. Switches are connected to each other via the bus coupled models. The 2×2 hierarchical component representation of this NoC hides the details of its parts (refer to [26] for the components and decompositions of the switch, network interface, and the bus model components). Note that any hierarchical model can be automatically flattened [27] and thus improve execution performance.

4. Experiments & results

In this section, our primary goal is to validate the NoC model developed in the previous section. In one set of experiments, we run these models in logical time and compare the outcome with BookSim under similar (but not identical) scenarios. In another set of experiments, we validate real-time execution capability and demonstrate the impact of processing power (of the execution platform) on the outcome of the simulation.

In the first phase (comparison with BookSim), we are well aware that this form of validation cannot explore the entire state space as only a finite set of simulation scenarios can practically be completed. These experiments are intended to provide a proof of concept and show how NoC models developed via DEVS can function similar to well-known simulators.

4.1. Comparison with booksim

We compare NoC simulation results in the extended DEVS-Suite simulator with their counterparts developed in BookSim for a selected number of network configurations. In order for the two to be comparable, we have simulated NoC in logical time since BookSim does not support real-time modeling.

BookSim uses a configuration file which holds all the necessary information for generating a NoC and simulating it. The input configuration file used in this experiment is presented in Listing 4. Throughout this section, we modify some of these parameters (injection rate and network size) to generate various NoC configurations.

The NoC models constructed in Section 3 are parameterized to generate different NoC configurations. Configuration elements of NoC in ALRT-DEVS are similar to those of BookSim. One needs to specify configurations such as routing algorithm, network size, topology, flow control mechanism, buffer size in ALRT-DEVS to specify a certain NoC. The one we used in this section uses on/off flow control method, X-Y routing algorithm, and round-robin allocation policy. The rest of the parameters

Table 6
Configuration variables of NoC (values in cycles).

Variable	Value	Variable	Value	Variable	Value
linkTraversalMin	1	swAllocationMax	1	numOfVirtualChannels	2
linkTraversalMax	1	swAllocationMin	1	maxInputBufferSize	8
statusTransmitterMax	1	swTraversalMax	1	numberOfPorts	4
statusTransmitterMin	1	swTraversalMin	1	numOfFlits	1
routingMax	1	vcAllocationMax	1		
routingMin	1	vcAllocationMin	1		

Table 7
The impact of injection rate on average flit latency in logical time (9-node mesh).

Generation rate (flit/cycle)	Average flit latency with DEVS-Suite (cycles)	Average flit latency with BookSim (cycles)	Difference	Difference %
0.10	15.506	15.40	+0.106	0.68%
0.13	15.548	15.70	−0.152	0.97%
0.15	15.840	15.98	−0.140	0.88%
0.18	16.484	16.80	−0.316	1.88%
0.20	16.782	17.44	−0.342	1.96%

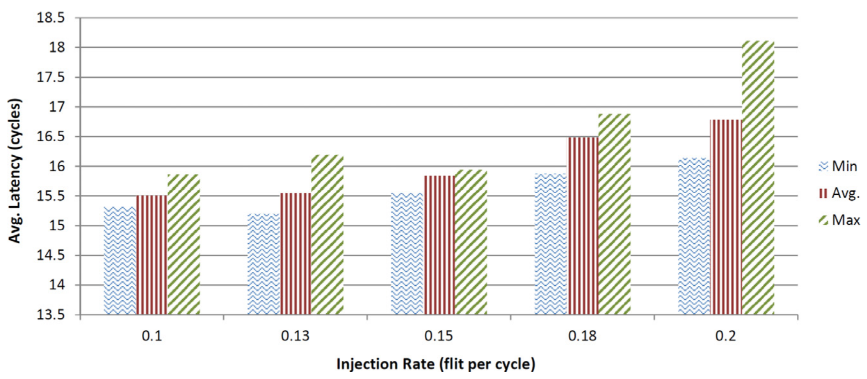


Fig. 6. Maximum, minimum, and average flit latency in logical time (experiment No. 1).

Table 8
The impact of network size on average flit latency in logical time.

Network size	Average flit latency with	Average flit latency with DEVS-Suite (cycles)	Difference BookSim (cycles)	Difference %
3 × 3	15.506	15.40	+0.106	0.68%
4 × 4	19.542	19.36	+0.182	0.93%
5 × 5	23.714	23.82	−0.106	0.45%
6 × 6	28.408	28.86	−0.452	1.59%

are presented in Table 6. For the experiments below, we changed the injection rate or the size of the network and observed their impacts on end-to-end average flit latency.

The models/simulations in this section are developed as proofs of concept. That is why we went for the simple case of 1 flit per packet. Since our routing algorithm routes flits separately, in order for the ALRT-DEVS model to be comparable with BookSim, we constrained packets to 1 flit. However, increasing this number is also about changing a configuration variable.

In a first experiment (No. 1), we analyzed the impact of injection rate on average flit latency by increasing the injection rate from 0.1 (flit/cycle) up to 0.2 (flit/cycle) step-by-step. Results are reported in Table 7 and Fig. 6. The bar chart provides a simple statistical analysis on five different runs. Randomness and different allocation policies are the main reasons for the divergence in the results observed from DEVS-Suite and BookSim (especially when the network is dealing with a heavier traffic flow of flits). This is evident in the difference % column in Table 7.

In a second experiment (No. 2), the impact of network size on the same variables is analyzed. The size of the network is increased from 9 to 36 nodes with the constant injection rate of 0.1 (flit/cycle). The results are reported in Table 8 and Fig. 7. The bar chart is again a statistical analysis of the simulated flit latency. As the network becomes congested (when flit buffers get full and the network performance drops significantly), the difference between the simulators and even runs of the same simulator are evident. This is due to the role which randomness plays in a congested network.

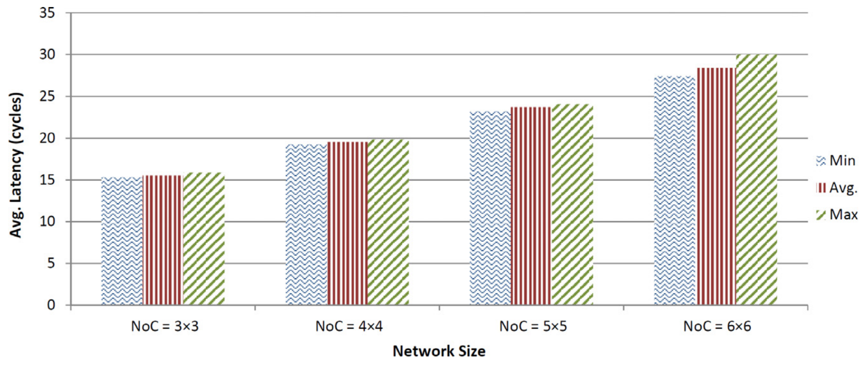


Fig. 7. Maximum, minimum, and average flit latency at 0.1 flit/cycle (experiment No. 2).

Table 9
NoC configuration.

Variable	Value	Variable	Value	Variable	Value
Virtual Channels	2	Num of Flits/Packet	1	SW Traversal	0.01 s
Buffer Size	8	Num of Ports	4	SW Allocation	0.01 s
Link Traversal	0.001 s	Routing	0.01 s		
VC Allocation	0.01 s	Status Traversal	0.01 s		

From these experiments, we can conclude that the DEVS-based and BookSim simulations of 3×3 NoC are comparable. Although there are larger differences in times of congestion, the two act very similar in regular traffic loads (for more details see [26]).

4.2. NoC example models

In order to show how the real-time simulator preserves the behavior of the model, platform saturation phenomenon is tested as one of the experiments. Although the model supports on/off flow control (which keeps flits at upstream switch when downstream buffers are full), in the real-time simulator, some actions may miss their deadlines which in turn undermines timeliness of the simulated experiments with respect to physical clock. Due to insufficient computational resources, some actions may not complete within designated time periods that are assigned to them by the model and enforced by the simulator. Therefore, any action missing its deadline should be repeated several times for each flit. This experiment is designed to explore the relation between the size of the target network (number of PEs or switches) and number of deadline misses in a 20-s period given the simulator's platform constraints.

Of course, insufficient computational resources only impact real-time executions in which simulation time must stay synchronized with physical time. However, the results in a logical time simulation in which time may go slower or faster, will not be affected by more powerful hardware.

In another experiment, we compare real-time and best effort executions of a $3\times$ 3 NoC model using real-time and logical-time simulators, respectively. This experiment compares the number of deadline misses in the hard real-time execution and soft real-time (as fast as possible) execution for the first 1500 flits delivered across packetizers, switches, buses, and de-packetizers. The reason we stopped at 1500 flits is that the network quickly reaches its steady-state; meaning that network performance metrics (such as queueing time or end-to-end average flit latency) reach a steady value. We repeated the experiments on average latency for a for a total of 3,000, 5,000, and 10,000 flits. The changes in the average latency were contained within a tenth of a clock cycle. For GHz clock frequencies this amount of change is negligible. This shows how the real-time execution protocol assists the simulation to stay synchronized with the simulator's real-time clock. There are various methods of data collection that can be used for these experiments: Experimental Frame [27], Flit-based Data Collection, and Periodic Reporting [28]. As for this experiment, data is collected from the network using the experimental frame method. This is because the number of events sent to the experimental frame is negligible as compared to those generated and consumed in the NoC itself.

We executed both of these experiments on a 32-bit Windows 7 computer with 2.93 GHz Intel Core 2 Due processor and 2GB of DDR2 memory with an effective maximum memory of 1.4GB. Also, Java 1.6 edition and Eclipse 4.2.0 are the execution and development platforms respectively. All models are realized and executed using DEVS-Suite 2.1.0 [8]. Table 9 specifies the timing behavior of the Model. All simulation data reported in this paper are the results obtained from five runs.

Table 10
Total actions executed and the number of deadline misses based on the size of the network for an injection rate of 10 (flits/s) for each processing element.

Num of nodes	Num of actions	Num of deadline misses	Ratio
4	2300	10	0.43%
9	6540	74	1.14%
16	14308	338	2.36%
20	20917	792	3.79%
25	26549	1334	5.03%
30	36001	3315	9.21%
36	43965	8383	19.07%
42	62303	29423	47.23%

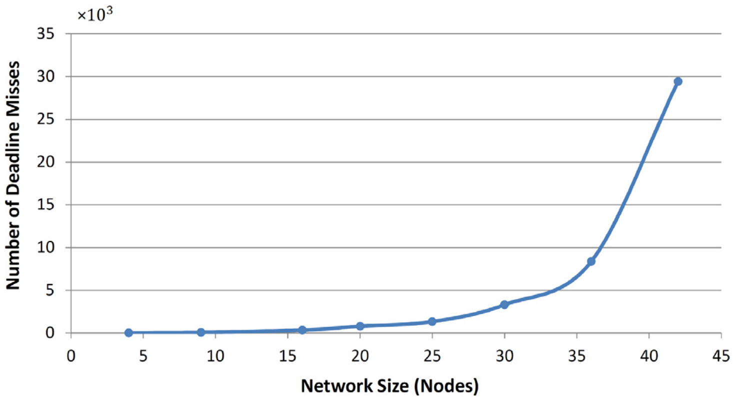


Fig. 8. Total actions executed and the number of deadline misses based on Table 10.

Table 11
Flit latency for hard real-time (ALRT-DEVS) and soft real-time executions with various injection rates .

Injection Rate (flits/sec)	ALRT-DEVS (sec)			Soft Real-time (sec)
	High	Low	AVG	AVG
10	0.169	0.168	.0168	0.498
12.5	0.192	0.182	.0186	0.559
16.67	0.489	0.416	0.458	0.809
25	2.617	2.256	2.515	1.073
30	3.543	3.3	3.404	2.957
40	4.811	5.421	4.597	16.258
50	5.271	5.119	5.197	20.480
60	6.420	5.901	6.161	44.474

4.3. Results & analysis

As Fig. 8 shows, the number of nodes and their connections are enlarged while a 20-s sample is taken for each configuration. In all configurations, the injection rate for individual processing elements are 10 flits per second. Samples count all actions executed and missed within this 20-s (see Fig. 8). There is a non-linear relation between the size of the network and the number of actions that missed their deadlines. The results demonstrate how the ratio of deadline losses to total number of actions increases when moving from smaller NoCs to larger ones. These results show quantitatively the error rate that can be expected from the simulation that executes in real-time given different network sizes. Due to the chosen simulation platform limitations, we were able to increase the network size to have up to 42 nodes. This limitation is due to the memory requirement which could not be met in 32-bit Java. Although stronger simulation platforms can support larger networks, this was not the focus in this experiment.

As for the second experiment, a 3×3 NoC was developed and observed under various injection rates (see Table 11). In this table, for each injection rate, the flit latency for hard real-time execution (ALRT-DEVS) and soft real-time execution are measured and reported. Soft real-time execution refers to conducting the simulation in best effort real-time but records the data using the time-stamps provided in the simulator. We ran each experiment in this Section 5 times to show that the impact of stochasticity in the model is negligible. Observing large variations in the results (excluding experiments that

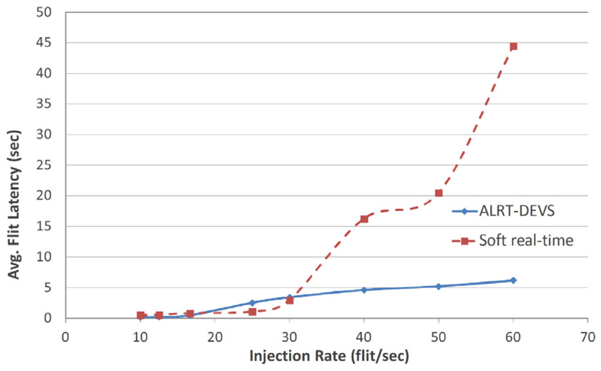


Fig. 9. Average flit latency for various injection rates.

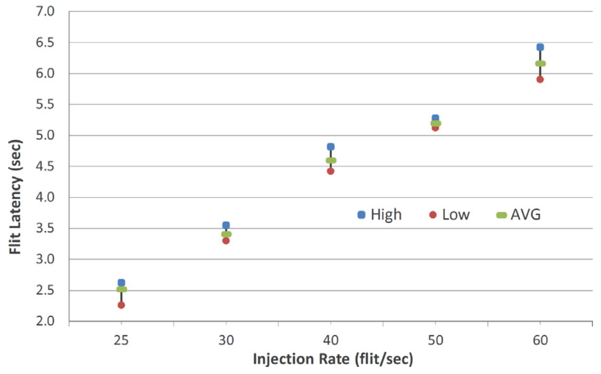


Fig. 10. Stock chart for 5 runs of each injection rate (Table 11).

exhibit saturated dynamics) is not considered to be acceptable for actual NoCs. We stopped at 5 repetitions as the results showed small variations with additional experiments. For the hard real-time execution, we reported the average, high, and low for 5 runs. As for the soft real-time, the average for all 5 runs is reported. In our experiment on average latency (reported as number of clock cycles), for the injection rate of 0.15 and sample size of 10,000 flits, the variance of the first 5 experiments were 0.00098 which is acceptable for a discrete event system with a granularity of one or half a clock cycle. As the plot in Fig. 9 shows, data retrieved from the NoC models demonstrates a linear growth while the injection rate is increasing. However, soft real-time shows a an exponential growth in average flit latency. This growing difference (between best effort and hard real-time) is the consequence of the real-time simulation protocol and the concurrency introduced in the simulation engine. Soft real-time DEVS-Suite uses the logical-time sequential execution protocol which is the root cause of its divergence from the hard real-time execution.

For these experiments we used average time instead of worst case analysis though our data collection and analysis method is capable of generating both. The worst case analysis is a necessity for NoC design as it decides weather the NoC is in compliance with functional requirements. The average case analysis on the other hand is meant for comparing the overall behavior of the system at various configurations. In addition to this, we used average case analysis for comparing extended DEVS-Suite with BookSim. The data gathered from experiments are statistically analyzed in Fig. 10. This analysis is based on the average, high, and low values reported in Table 11 for all injection rates in experiment No. 2. Since the difference between maximum, minimum, and average is negligible for the first 3 rows of data, we have excluded them from Fig. 10. As shown, the difference grows as we increase the traffic beyond the saturation point of 16.67 flits/s (see also Fig. 8).

Before moving on to the next experiment, it is useful to note that computing platforms, not the modeling approach, can adversely affect real-time simulations [29]. As an example, we simulated one experiment on two different computing platforms. The injection rate is set at 2 flits/s and the size of the network is gradually increased. As shown in Table 12, a platform with higher computing power can simulate larger size networks at higher accuracy as measured in terms of fewer deadline misses as well as reduction in missed deadlines. However, every computing platform eventually fails to be useful as the network size is increased (e.g., 10×10 network). It is worth mentioning that both platforms are capable of simulating, in logical-time, larger NoC systems (up to the memory limit).

In order to illustrate the difference in the operation of real-time and soft real-time modeling and simulation, several flits are closely monitored. These flits are randomly chosen and tagged. In Fig. 11, injection rates 12.5, 25, 33, 40, 50, and 62.5 (flits/s) are observed with a sample of 15 to 20 flits traveling from node 0 to node 8 (the diagonal path). The horizontal axis specifies the injection cycle of the flit and the vertical axis demonstrates the total latency. Fig. 11a depicts flit latencies

Table 12
Impact of computing platforms on simulating large NoC systems .

	Dual Core 2.9GHz, 2GB Mem.			Quad Core Xeon 3.7GHz, 16GB Mem.		
NoC size	Total actions	Deadline misses	Error percentage	Total actions	Deadline misses	Error percentage
5 × 5	21435	546	2.55%	21292	98	0.46%
6 × 6	34567	1399	4.05%	34262	550	1.61%
7 × 7	54451	3512	6.45%	54056	812	1.50%
8 × 8	–	–	–	78024	2017	2.59%
9 × 9	–	–	–	109039	6046	5.54%
10 × 10	–	–	–	191442	60035	31.36%
11 × 11	–	–	–	304402	123532	40.58%

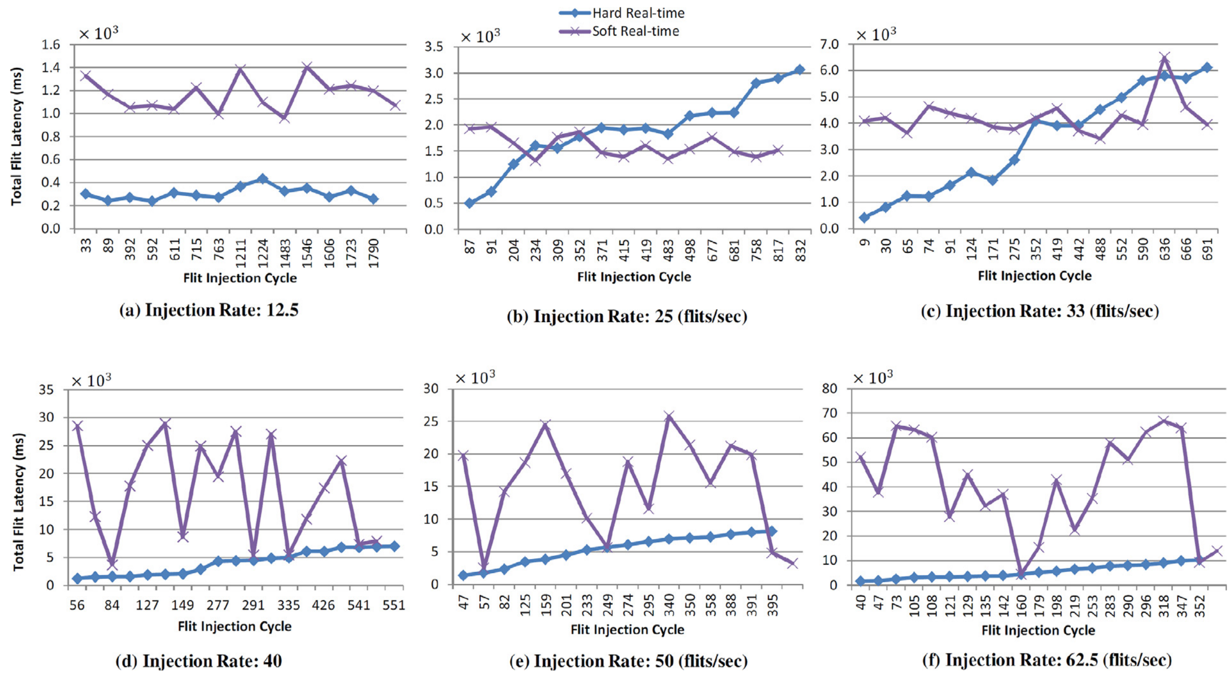


Fig. 11. Relation between time of injection and total latency for flits traveling from node 0 to node 8 in experiment II with various injection rates: (a) logical-time execution protocol is the dominant overhead, (b) real-time threading overhead causes latency to grow, (c) network becomes more congested and the number of parallel events results in poorer performance from soft real-time, (d-f) the network is congested and soft real-time shows significant divergence from the real value.

for a non-congested network in which queue overflow or long waiting does not occur. As we increase the injection rate, increase in the flit latency is evident in Fig. 11b and c. The real-time execution engine demonstrates a linear increase in the latency of the flits later injected in the network. For high injection rates a gradual increase in the total latency of the flits is expected (see Fig. 11d–f). The NoC execution demonstrates a linear increase of latency for the flits injected later in the simulation. However, significant variations are evident in the execution of soft real-time model in which the overhead of logical execution dominates other factors (even the congestion of the network). If a flit is serviced ahead of others (since the execution protocol is sequential), its latency will be closer to the NoC, but if it has to stay behind a queue of other flits to be handled, the latency would be significantly higher (higher waiting times). This phenomena becomes more severe when the network traffic increases as the difference between soft real-time and real-time becomes greater with higher packet injection rates.

Another way that we can compare hard and soft real-time executions in DEVS-Suite is by looking at individual actions, their release time, and their deadlines. For this, we look at the stages a flit goes through in the switch component from the time it is received at the input port until it is sent over the output port to be transmitted out. Fig. 12a demonstrates the timeline of various stages the flit undergoes from entering the input port until leaving it. As the network gets more congested, it is expected to see longer waiting times. Also, because of the high contention between flits to allocate the virtual channels and crossbar switch, multiple virtual channel allocation or switch allocation attempts may be required which will elongate the waiting time even further. This is also shown for the model in Fig. 12c and the same phenomenon (longer queuing times and multiple allocation attempts) can be observed. Fig. 12b shows the timeline of actions executed by the input port component (modeled in DEVS-based specification). Deadline violations in the timeline are shown with

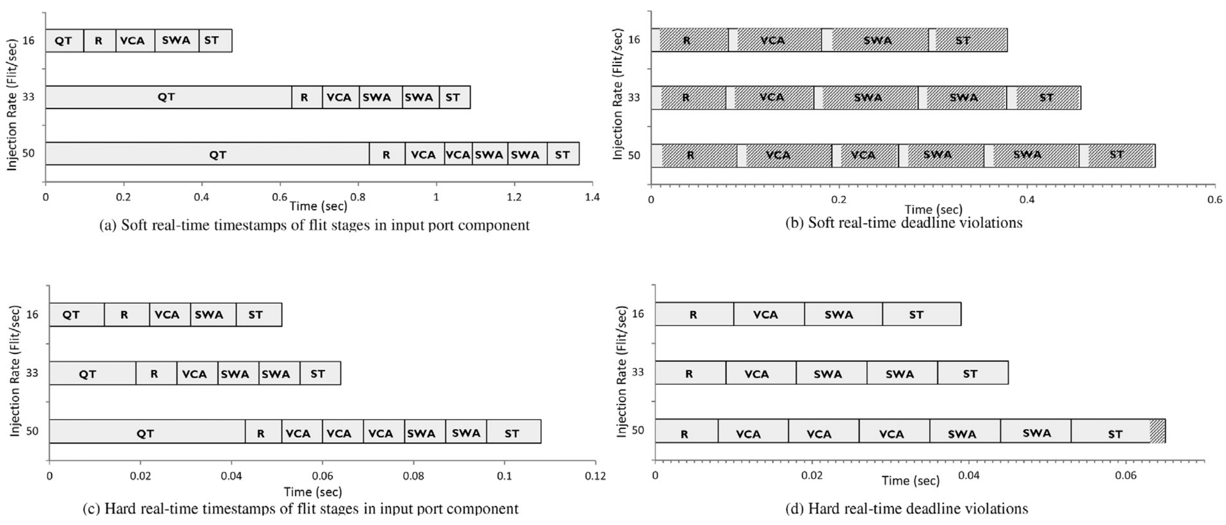


Fig. 12. Timelines of flit stages in input port and actions executed by the input port with various injection rates for hard and soft real-time (QT: Queueing Time, R: Routing, VCA: Virtual Channel Allocation, SWA: Switch Allocation, ST: Switch Traversal).

diagonal lines (soft real-time does not guarantee timeliness of action execution). Fig. 12d depicts the same diagram for NoC action execution. In this run, only one action has violated its deadline (for 50 flits per second injection rate). These diagrams show the difference between action-level, real-time modeling and simulation with that of logical-time DEVS at the level of actions (higher levels were analyzed in previous experiments).

Recognizing the distinction between the model and the simulator is essential when dealing with situations such as network saturation. The impact of increasing network size or increasing flit injection rates on experiment results (such as total flit latency in experiment no. 2) has two sources, one originating in the model and the other in the simulator:

- Model: increasing the size of the network (or higher injection rate) means additional flit traffic and collisions. This in turn can contribute to longer waiting times and consequently higher total flit latency.
- Simulation: the number of concurrent components running on the simulation platform can increase rapidly as the network's size increases. From a certain point forward, the load cannot be handled properly by the simulation platform and the number of deadline losses increases (which would not have happened if logical time simulator is used). This results in higher delay for one flit to reach its destination which in turn increases the average flit latency.

Different kinds of analysis of the simulation results become necessary when working with real-time simulation. The simulation platform, the system under simulation, data collection method, time granularity, and communication with the environment (which the executing platform is a part of) are all contributing factors beside the model and the input data (i.e., designs and configurations for the processing elements).

5. Related work

Modeling languages such as DEVS and Petri net are used for computer hardware systems. Logical-time models for single-cycle, multi-cycle, and pipeline MIPS32 are developed using DEVS and simulated using the DEVS-Suite simulator [25]. Also, embedded system design and HW/SW-codesign using DEVS model of computation has been suggested in [30,31]. An approach for modeling hardware systems with DEVS and HLA simulation, in logical-time, is also proposed in [32]. Concentrating on NoC modeling, colored Petri net is used for detailed NoC modeling [33]. This model is then used for verifying and analyzing communication protocols in NoC. Similar works using Petri nets can be found in [34–36]. Unlike Petri net, the NoC models in this paper are defined in terms of time-constrained actions. As for DEVS-based modeling, a parallel logical-time DEVS model for NoC is developed [37] and implemented in DEVS-Suite simulator for performance analysis. Several well-known topologies with various routing policies are implemented and the results are compared with Noxim [38]. In [12], the concept of multiresolution modeling was adopted and models of NoC were created in various resolutions. However, the focus of that work is on defining resolution (with respect to object, time, process, and spatial aspects) for NoC; in addition, all models are created and simulated in logical-time.

Because of the importance of simulation in early stages of NoC design, numerous simulators have been developed in that past several years for analyzing Network-on-Chips. Among these simulators, some operate on lower levels (Register Transfer Level) which provide accurate analysis on energy consumption, area, and performance [39]. GEM5 [40] is the combination of M5 [41] and GEMS [42] simulators. M5 provides diverse CPU and ISA (Instruction Set Architecture) simulations, which coupled with memory system and cache coherency simulation capabilities of GEMS results in a complete and accurate system simulation of GEM5. Simulation with this level of accuracy only supports a few system combinations. This is due to the unavailability of models for different brands of CPU, bus, and memory.

Several other simulators support higher levels of system abstraction in which results are less accurate but instead they are retrieved more efficiently. In this group of simulators, BookSim [10] is one of the most popular ones in academia. BookSim is implemented in C++ with high level of expressiveness. Also, it is possible to couple this simulator with energy and area estimators to support power and area analysis. Noxim [38] is another simple and expressive simulator implemented based on SystemC. Noxim simulator is shipped by a complementary product called Noxim Explorer which explores the state space by changing simulation parameters automatically.

The approach devised in this paper clearly requires more details when it comes to specifying NoC models. DEVS formalism supports modeling a wide range of systems while specialized environments such as BookSim and Noxim are optimized for NoC modeling: they have brief and concise syntaxes for specifying NoC models. Similarly, DEVS-Suite simulation engine (even in logical-time mode) is considerably slower in executing NoC models. However, as discussed earlier, environments based on explicit formalisms are more flexible and extendable. This DEVS-based approach can be extended to support model checking and multiresolution modeling in addition to the already existing simulation capabilities [11].

In contrast to BookSim and Noxim simulators, DARSIM [43] is more concerned with parallelization of simulation engine. However, DARSIM deals with the trade off between performance (which is higher for fewer synchronizations) and accuracy (lower for higher performance) of analysis results. This simulator is suitable for high-level many-core NoC simulations for which parallelization contributes to less execution time. This simulator uses multi-core and hyper-threading capabilities of CPU for parallel execution of threads; therefore, gaining better performance.

Nostrum [44] is another Network-on-Chip simulator which is primarily concerned with the communication issues between the physical layer and the application layer. This simulator is capable of applying application-oriented traffic to the model as well as synthetic traffic. It conducts the simulation at flit-level cycle-by-cycle in order to offer more accurate results, which in this case is close to BookSim and Noxim introduced above. TOPAZ [45] provides a more detailed model of NoC by focusing on multicasting, router pipeline, and various message types. This simulator can be standalone or used with GEMS or GEM5. In addition, this simulator has support for parallel execution using POSIX threads.

ATLAS [46,47] is a framework, providing a unified process: from NoC generation to power evaluation. It receives information such as topology, routing algorithm, flow control mechanism, virtual channels, and bandwidth from the user to generate NoC in VHDL. Later a traffic generation module produces various traffic patterns to be tested on the NoC. Simulation is done by another tool: ModelSim, which provides raw data for performance evaluation. Performance evaluator calculates several standard network measures such as average delivery time. Finally, power analysis is done based on injection rate at each port.

In Register Transfer Level (RTL) simulation, the simulator is involved in low-level interactions between components which results in longer execution time. For this reason, some simulators are synthesized on FPGA to provide faster execution while maintaining the same level of accuracy. Dart [6] is an FPGA-based NoC simulation engine which provides considerable performance gain over common simulators (others mentioned in this section). It is possible to integrate this simulator with full system simulators to support real application or trace-driven traffic instead of synthetic traffic. However, Dart does not provide much support for various configurations of routers. There are limited options for flow control mechanism and routing algorithms which result in accurate but limited configurations of NoC.

6. Conclusion

In this paper we developed a new NoC model at flit-level using the Action-Level, Real-Time Discrete-event System Specification modeling and simulation approach. DEVS-Suite simulator (with its extension for real-time execution) is developed to simulate these NoC models in logical-time and soft/hard real-time. Experiments show that logical-time NoC simulator offers modeling capabilities that achieve accuracy similar to the BookSim and Noxim NoC simulation platforms.

As detailed in Section 5, there exists many simulators targeted for developing and evaluating different levels of NoC abstraction. However, an important question is: given existing RTL simulators (such as GEM5), what could be the advantages of higher level simulators? To answer this question, we refer to the concept of multiresolution modeling. Multiresolution modeling suggests an incremental process which facilitates moving from very high-level models (without implementation details) to RTL models ready to be implemented in hardware. Each level of abstraction introduces more details to the abstraction level above it. As we move upward in model abstraction hierarchy (i.e., removing details), the resulting model becomes smaller from the point of view of state-space size, faster execution time (for validation or verification), and easier to optimize (as done in [48]) and analyze. The proposed NoC ALRT-DEVS models are well-suited for making decisions such as the topology of the network, number of cores, number of virtual channels, switch architecture, and arbitration policy.

Simulators introduced in Section 5 are mostly time-accurate communication models in which timing and protocols are modeled for communication but computation time and processing element interfaces are not accounted for in any great detail. Although the NoC model developed in this paper also falls into the same category (as shown in our comparison with BookSim), it is distinguishable from the rest due to its direct support for specifying actions with time windows. This approach is positioned to allow action-level, real-time NoC simulations to communicate with peripheral hardware. The DEVS-Suite's I/O ports are extended such that DEVS models can interact interchangeably with software, hardware, or other simulations [19].

Acknowledgement

We acknowledge helpful comments from anonymous referees, Dr. Umit Ogras, and Dr. Aviral Shrivastava at Arizona State University on an earlier version of this paper.

References

[1] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, D. Lindqvist, Network on chip: an architecture for billion transistor era, in: Proceeding of the IEEE NorChip Conference, 31, 2000.
[2] V. Rantala, T. Lehtonen, J. Plosila, Network on chip routing algorithms, Citeseer, 2006.
[3] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, Y. Hoskote, Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives, Comput.-Aided Design Integr. Circuits Syst., IEEE Trans. 28 (1) (2009) 3–21.
[4] J. Zarrin, R.L. Aguiar, J.P. Barraca, Manycore simulation for peta-scale system design: motivation, tools, challenges and prospects, Simul. Modell. Pract. Theory 72 (2017) 168–201.
[5] W. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann, 2004.
[6] D. Wang, N.E. Jerger, J.G. Steffan, DART: a programmable architecture for NoC simulation on FPGAs, in: Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip, ACM, New York, NY, USA, 2011, pp. 145–152. URL http://doi.acm.org/10.1145/1999946.1999970.
[7] H.S. Sarjoughian, S. Gholami, Action-level real-time devs modeling and simulation, Simulation (2015). 0037549715604720.
[8] ACIMS, DEVS-suite simulator, version 3.0.0, 2015, (https://acims.asu.edu/software/devs-suite/).
[9] S. Kim, H.S. Sarjoughian, V. Elamvazhuthi, DEVS-Suite: a simulator supporting visual experimentation design and behavior monitoring, in: Proceedings of the 2009 Spring Simulation Conference, Society for Computer Simulation International, 2009, pp. 29–36.
[10] N. Jiang, J. Balfour, D.U. Becker, B. Towles, W.J. Dally, G. Michelogiannakis, J. Kim, A detailed and flexible cycle-accurate network-on-chip simulator, in: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on, IEEE, 2013, pp. 86–96.
[11] S. Gholami, H.S. Sarjoughian, Modeling and verificaiton of network-on-chip models using constrained-DEVS, in: Proceedings of the 2017 Spring Simulation Multiconference, Society for Computer Simulation International, 2017, pp. 1–12.
[12] S. Gholami, H.S. Sarjoughian, Multi-resolution co-design modeling: a network-on-chip model, in: Proceedings of the 2016 Winter Simulation Conference, IEEE Press, 2016, pp. 1499–1510.
[13] S. Gholami, H.S. Sarjoughian, Real-time network-on-chip simulation modeling, in: SIMUTools, Desenzano, Italy, ICST, 2012, pp. 103–112.
[14] J. Hong, H. Song, T. Kim, K. Park, A real-time discrete event system specification formalism for seamless real-time software development, Discrete Event Dyn. Syst. 7 (4) (1997) 355–375.
[15] Q. Wang, F.E. CELLIER, Time windows: automated abstraction of continuous-time models into discrete-event models in high autonomy systems? Int. J. Gener. Syst. 19 (3) (1991) 241–262.
[16] H. Giese, S. Burmester, Real-time statechart semantics, TechReport tr-ri-03-239, University of Paderborn, 2003.
[17] A. Chow, B.P. Zeigler, Parallel DEVS: A parallel, hierarchical, modular, modeling formalism, in: Proceedings of the 26th conference on Winter simulation, Society for Computer Simulation International, 1994, pp. 716–722.
[18] R.M. Fujimoto, Parallel and Distributed Simulation Systems, Wiley, 2000.
[19] H.S. Sarjoughian, S. Gholami, T. Jackson, Interacting real-time simulation models and reactive computational-physical systems, in: Proceedings of the 2013 Winter Simulation Conference: Simulation: Making Decisions in a Complex World, IEEE Press, 2013, pp. 1120–1131.
[20] J. Spars, S. Furber, Principles Asynchronous Circuit Design, Springer, 2002.
[21] A.J. Martin, S.M. Burns, T.-K. Lee, D. Borkovic, P.J. Hazewindus, The first aysnchronous microprocessor: the test results, SIGARCH Comput. Archit. News 17 (4) (1989) 95–98. URL http://doi.acm.org/10.1145/71317.71324.
[22] EPSON, ACT11, an 8-bit asynchronous chip, 2004.
[23] S.M. Nowick, M. Singh, High-performance asynchronous pipelines: an overview, Des. Test Comput., IEEE 28 (5) (2011) 8–22.
[24] D.M. Chapiro, Globally-asynchronous locally-synchronous systems, Stanford University, CA., 1984 Ph.D. thesis.
[25] Y. Chen, H.S. Sarjoughian, A component-based simulator for MIPS32 processors, Simulation 86 (5-6) (2010) 271–290.
[26] S. Gholami, H.S. Sarjoughian, Network-on-Chip (NoC) Simulation with ALRT-DEVS: Supplementary Material and Experiments, Technical Report, Arizona State University, 2013. URL http://acims.asu.edu/wp-content/uploads/2014/02/ASUCIDSE-CSE-2013-001.pdf.
[27] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, 2nd, Academic Press, Inc., Orlando, FL, USA, 2000.
[28] S. Gholami, H.S. Sarjoughian, Observations on real-time simulation design and experimentation, in: Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium, in: DEVS 13, Society for Computer Simulation International, San Diego, CA, USA, 2013, pp. 15:1–15:8. URL http://dl.acm.org/citation.cfm?id=2499634.2499649
[29] A. Benveniste, L.P. Carloni, P. Caspi, A.L. Sangiovanni-Vincentelli, Heterogeneous reactive systems modeling and correct-by-construction deployment, in: Embedded Software, Springer, 2003, pp. 35–50.
[30] H.G. Molter, SynDEVS Co-Design Flow: A Hardware/Software Co-Design Flow Based on the Discrete Event System Specification Model of Computation, Springer, 2012.
[31] H.G. Molter, S. Huss, The DEVS model of computation a foundation for a novel embedded systems design methodology, in: International Conference on Computer Engineering Systems, 2011, pp. 21–26, doi:10.1109/ICCES.2011.6140989.
[32] A. Saghir, T. Pearce, G. Wainer, Modeling computer hardware platforms using DEVS and HLA simulation, in: 2004 Summer Simulation Conference, San Jose, California, 2004.
[33] H. Bazzaz, M. Sirjani, R. Khosravi, S. Taheri, Modeling networking issues of network-on-chip: a coloured Petri Nets approach, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST, 2009, pp. 22–32.
[34] H. Blume, T. von Sydow, D. Becker, T. Noll, Application of deterministic and stochastic petri-Nets for performance modeling of NoC architectures, J. Syst. Archit. 53 (8) (2007) 466–476.
[35] H. Blume, T. von Sydow, D. Becker, T. Noll, Modeling NoC architectures by means of deterministic and stochastic petri nets, Embedded Comput. Syst. (2005) 374–383.
[36] N. Chaki, S. Bhattacharya, Performance analysis of multistage interconnection networks with a new high-level net model, J. Syst. Archit. 52 (1) (2006) 56–70.
[37] H. Ahmadinejad, F. Refan, H.S. Sarjoughian, NoC simulation modeling in DEVS-Suite, in: Spring Simulation Conference, Orlando, FL, ACM Press, 2011, pp. 134–139.
[38] V. Catania, A. Mineo, S. Monteleone, M. Palesi, D. Patti, Cycle-accurate network on chip simulation with noxim, ACM Trans. Model. Comput. Simul. 27 (1) (2016) 4:1–4:25.
[39] H. Wang, X. Zhu, L. Peh, S. Malik, Orion: a power-performance simulator for interconnection networks, in: Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002.
[40] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, D.A. Wood, The GEM5 simulator, SIGARCH Comput. Archit. News 39 (2011) 1–7. http://doi.acm.org/10.1145/2024716.2024718.
[41] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, S. Reinhardt, The M5 simulator: modeling networked systems, Micro, IEEE 26 (4) (2006) 52–60.
[42] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, D. Wood, Multifacet'S general execution-driven multiprocessor simulator (GEMS) toolset, ACM SIGARCH Comput. Architect. News 33 (4) (2005) 92–99.

[43] M. Lis, K. Shim, M. Cho, P. Ren, O. Khan, S. Devadas, DARSIM: a parallel cycle-level NoC simulator, MoBS 2010 - Sixth Annual Workshop on Modeling, Benchmarking, and Simulation, 2010.
[44] Z. Lu, R. Thid, M. Millberg, E. Nilsson, A. Jantsch, NNSE: nostrum network-on-chip simulation environment, in: Swedish System-on-Chip Conference, April, 2005, pp. 1–4.
[45] P. Abad, P. Prieto, L.G. Menezo, A. Colaso, V. Puente, J.A. Gregorio, TOPAZ: An open-source interconnection network simulator for chip multiprocessors and supercomputers, IEEE Computer Society, Los Alamitos, CA, USA, 2012, pp. 99–106. http://doi.ieeecomputersociety.org/10.1109/NOCS.2012.19.
[46] A. Mello, N. Calazans, F. Moraes, ATLAS-an environment for NoC generation and evaluation, Design Automation and Test in Europe (DATE), 2011. URL https://www.date-conference.com/files/file/date11/ubooth/125.pdf.
[47] Hardware Design Support Group (GAPH), ATLAS environment, 2011. URL https://corfu.pucrs.br/redmine/projects/atlas.
[48] T.J. Pifer, DEVS-Based Hardware Design, Synthesis, and Power Optimization Using Explicit Time Specifications and Deterministic Path-Based Latency, University of Arizona, 2012 Master's thesis.