IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 December 2022, accepted 28 December 2022, date of publication 9 January 2023, date of current version 17 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3235276

RESEARCH ARTICLE

A Top-Down Modeling Approach for Networks-on-Chip Components Design: A Switch as Case Study

V. A. DELGADO-GALLARDO[1], (Member, IEEE), R. SANDOVAL-ARECHIGA[2], (Member, IEEE), AND R. PARRA-MICHEL[1], (Member, IEEE)

[1]Department of Electrical Engineering, CINVESTAV, Zapopan 45017, México
[2]Centro de Investigación, Innovación y Desarrollo en Telecomunicaciones (CIDTE), Universidad Autónoma de Zacatecas, Zacatecas 98060, México

Corresponding author: R. Parra-Michel (ramon.parra@cinvestav.mx)

This work was supported by the Consejo Nacional de Ciencia y Tecnología (CONACyT) under Grant 718317.

ABSTRACT The design of Networks-on-Chip (NoCs) components implies a wide range of techniques and methods to address the microarchitecture of the packet-forwarding components, where routers and switches are the most complex because they constitute the NoC's backbone. Due to this complex design space, several works use approaches limiting architectural exploration, focusing only on achieving high-performance levels; therefore, they are inadequate for designing NoC components when particular functionalities are demanded, as in real applications with specific protocols and interfaces. This paper presents a design methodology based on a top-down approach with NoC-oriented abstraction levels to systematically generate a microarchitecture and its hardware description according to system requirements. The design flow transforms a high-level functional model into a microarchitecture model through a refinement process at each abstraction level. This structured approach involves integrating details on how the data is functionally managed within the component according to the system requirements and the processing granularity of each level, allowing testing alternatives in the early stages of the design when necessary. The models of each abstraction level can be described and simulated using the simulator OMNet++. Thus, the obtained microarchitecture model will be directly translated into a Hardware Description Language (HDL). The methodology is tested via the design of a NoC switch for a Software Defined Radio (SDR) system. Performance analysis and implementation results in a field-programmable gate array (FPGA) show that the proposed design is functional and comparable in both area and frequency to other similar state-of-the-art components, and it is also configurable to build star topologies of up to 16 nodes.

INDEX TERMS Design methodology, microarchitecture, modeling techniques, Networks-on-Chip, switch component.

I. INTRODUCTION

Current silicon technologies allow Systems-on-Chip (SoCs) to have many modules such as CPUs, memories, Intellectual Property (IP) cores, etc. Because of this growth, Networks-on-Chip (NoCs) have been widely adopted as a new on-chip interconnection paradigm to meet recent and future parallel processing applications demands of the industry [1], [2], [3], [4]. An important research topic regarding on-chip networks is the design and implementation of the NoC components, where the routers and switches are the most complex designs.

The microarchitecture design of these components involves a vast design space, where a set of techniques and well-accepted practical solutions are used to implement aspects of the communication paradigm such as flow

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

control, routing, arbitration, buffering, and quality of service [5], [6], [7], [8], [9].

Consequently, different strategies are used for router and switch microarchitecture design. In accordance with a recent literature review, a taxonomy of router[1] design approaches is suggested and discussed below. See Table 1.

A. DESIGNS BASED ON CANONICAL ARCHITECTURE MODIFICATIONS

Router design approaches based on canonical architecture modifications use a model as a reference, mainly the model proposed by Peh and Dally [10], which introduces the basic wormhole and virtual-channel (VC) router architectures. By using these base models, specific architectural design areas can be studied and improved.

1) DATAPATH DESIGN

It focuses on how to implement the basic components of a router and datapath enhancements to support novel data transport techniques in the NoC. Previous works address logic implementation techniques [6], [7], reliability techniques [11], [12], [13], [14], implementations in CMOS process [15], [16], broadcast and multicast support [17], [18], high-radix router design [19], multiple injection and ejection ports scheme [20], novel switching schemes [21], time-multiplexing schemes for VC [22] or even integration of machine learning techniques for datapath design [23], [24], [25], [26].

2) PIPELINE DESIGN

The analysis of how to improve the router pipeline stages is studied in this area to get high-speed designs. There are approaches to reduce the router latency to a single cycle using speculative [27], [28], non-speculative [29], and prediction strategies [30], and also the use of pipeline bypassing techniques can achieve latency reduction [31], [32].

3) LINK DESIGN

Architectural and circuit-level techniques at the NoC links are used to improve performance and save chip area. Examples of these approaches are timing-error-tolerant [33], elastic buffers [34], [35], [36], dual-function links [37], double-data-rate links [38] and bidirectional links [5].

4) MEMORY ORGANIZATION DESIGN

This area covers memory management techniques to optimize buffers. The most common techniques are central memory, individual buffers, or schemes for VCs support [6], [17], [34], [37], [39], [40], [41], [42], [43].

5) LOW POWER DESIGN

This design area is focused on techniques and methodologies for energy-efficient architectures [44], [45], [46], [47].

[1]The proposed taxonomy is considered for router and switch design. However, most research work focuses on router design because functionally, a router is a switch with a routing stage to determine the path of the packets.

B. AD HOC APPROACHES

In the ad hoc approaches, several works generate architectures alternative to canonical designs, proposing new strategies for managing and handling the packets within the router to improve NoC performance.

There are modular designs where the router internally handles each dimension of the NoC independently [54], [55], [56], [57], [58], [59], shared-buffers approaches [60], [61], [62], path-sensitive with decomposed crossbar designs [63], [64], [65], bufferless approaches [70], [71], [72], and other design alternatives such as decentralized structure based on rings [66], pre-configured paths [67], hybrid packet/circuit switching [68], and crossbar folding with marching memory buffers [69].

C. LIBRARY-BASED APPROACHES

Instead of improving a baseline design or proposing new ones, the library-based approaches use tools, frameworks, or strategies to automate a design flow like the work in [73]. For router design, tools are used to tune the parameters of a router architecture composed of a predetermined library of building blocks [48], [49], [50]. Some works even include the hardware synthesis of the entire NoC [9], [51], [52], [53]. These approaches aim to use high-level specifications such as inter-core communication demands and design goals to optimize performance.

D. MOTIVATION FOR A NEW DESIGN METHODOLOGY

From a methodological point of view, the common approach for router design is to limit the architectural design space to a few aspects through a baseline model (fixed architecture) or a configurable library-based model (flexible architecture). In fact, the architectures provided by these methods were designed only to support generic protocols. All of this makes sense because most research works focus on improving performance. However, these approaches are inadequate for dedicated NoC components design, i.e., systems requiring to comply with particular NoC functionalities such as specific data-link protocols, packet formats, interfaces, or configurable features. Examples of these systems are FCUDA-NoC [53], VBON [18], and the NoC for the TeraFLOPS processor [16].

The last two columns of Table 1 summarize how architectural exploration is initiated and carried out when particular functional requirements need to be met. Using approaches based on canonical architectures provides a well-understood and verified architecture. However, the canonical model was designed to use wormhole with VCs and credit-based link-level flow control. Therefore, the main drawback arises in deciding which architecture blocks should be modified, added, or even removed to comply with the functional requirements.

The library-based approaches provide a flexible architecture; therefore, the architectural exploration will depend on the available building blocks and their configurability

TABLE 1. Taxonomy of the design approaches for routers and an overview of how their architectural exploration is carried out to meet particular functional requirements.

NoC Router Design Approaches		References	How Architectural Exploration Begins	How Functional Requirements are Met
Based on canonical architecture modifications	Datapath Design	[6], [7], [11]–[26]	With a fixed architecture	Architecture blocks are modified, added, or even removed
	Pipeline Design	[27]–[32]		
	Link Design	[5], [33]–[38]		
	Memory Organization Design	[6], [17], [34], [37], [39]–[43]		
	Low Power Design	[44]–[47]		
Library-based approaches		[9], [48]–[53]	With a flexible architecture	It depends on the level of configurability of the available building blocks
Ad hoc approaches		[54]–[72]	No dependency on fixed or flexible architectures	It depends on the designer's experience and creativity

level. For example, the switch from the *xpipes* library [74] is highly configurable regarding the I/O ports, topology (because it uses source routing), the number of VCs, and the link buffer size, but its switching (wormhole) and arbitration (round-robin) schemes are fixed. Thus, the required specific functionalities might not be met, forcing changing or modifying the initial functional requirements, which is not desirable. Also, license costs and third-party dependency should be considered.

Finally, although the ad hoc approaches do not depend on baseline designs and achieve good performance results, they lack a design methodology. There is a certain degree of freedom to explore architectural alternatives in the design space for specific requirements. However, the novel architectures rely on the designer's insight.

For these reasons, it would be ideal to have a suitable approach for dedicated NoC components design that does not rely on canonical architectures. It is also desirable to rely on a strategy that allows addressing the architectural exploration to meet a given NoC protocol and required customizations.

E. SUMMARY AND CONTRIBUTIONS

In this work, a methodology for dedicated NoC components design and implementation is proposed. This approach is based on a detailed adaptation of the top-down modeling to be oriented to the NoC paradigm. Consequently, different levels of abstraction are proposed, allowing the designer to focus only on the component key functionalities according to each level without worrying about other implementation details, which the designer will add later.

The methodology begins with an abstract functional model at the highest level. Then, this model will be iteratively transformed into a new version with new functional design aspects according to the next abstraction level. Hence, the designer will decide which component functionality, such as flow control strategy, routing algorithm, switching technique, or microarchitecture composition, will be addressed according to the given NoC protocol and level of granularity. In this way, architectural exploration is based on the functional requirements, progressively satisfied because they are added and detailed according to each abstraction level. In addition, the proper functionality of the component is validated during the component design process. Finally, the microarchitecture model can be translated directly into a Hardware Description Language (HDL). A switch intended to build star topologies for a Software Defined Radio (SDR) system is used as an example of the results that can be obtained using the proposed methodology.

The main contributions of this work are:

- The study and analysis of NoC router design approaches and their generalities related to the architectural exploration to comply with specific functional requirements. Although many efforts to improve NoC performance involve various well-accepted techniques and practical solutions having provided, however, these approaches do not specialize in designing and implementing dedicated routers without relying on fixed and flexible architectures that limit architectural design space exploration.
- A methodology for designing and implementing NoC components such as switches and routers according to their functional requirements, which are systematically addressed in a top-down fashion. NoC-oriented levels of abstraction are proposed to allow exploring the architectural design space of the component in a structured manner.
- The design and implementation of a switch component for an SDR system using the proposed methodology. The switch performance using traces of real workloads and synthetic traffic patterns to analyze its operating limits is also included. Furthermore, an evaluation of area and frequency shows that the implemented switch is comparable to other similar state-of-the-art components designed using the traditional approaches.

The remainder of this paper is structured as follows: section II describes the functional requirements for a NoC switch component for an SDR system, such as the one described in [75]; section III describes the proposed methodology; section IV presents the experimental case of the study based on the requirements presented in Section II;

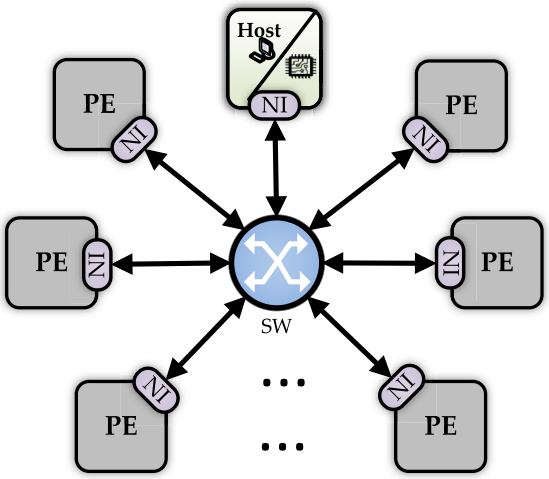


FIGURE 1. NoC basic architecture based on a switch. A set of reconfigurable PEs communicates simultaneously with each other to generate a multi-standard architecture. The host is responsible for tasks management.

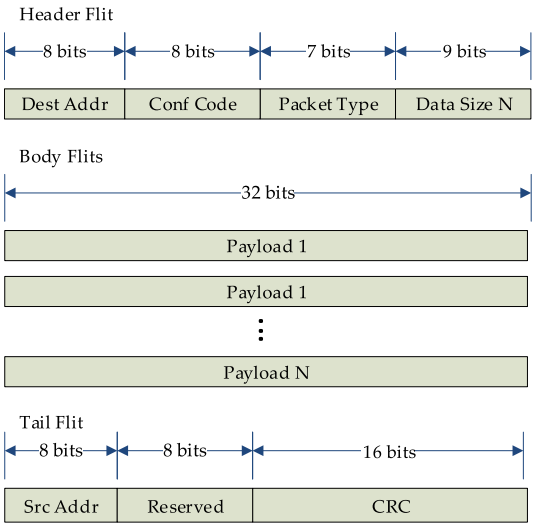


FIGURE 2. Packet structure required for the NoC of the SDR system.

section V shows implementation results and a comparison with literature approaches; finally, section VI presents the conclusions obtained in this paper.

II. REQUIREMENTS FOR AN SDR NoC SWITCH

In this section, a set of specifications and requirements for a NoC switch component to create hardware/software applications for digital communication systems under the SDR concept is presented in order to discuss the challenges when designing these kinds of components.

The required SDR system addresses multi-core SoCs composed of heterogeneous processing elements (PEs) containing reconfigurable signal processing (SP) algorithms, e.g., channel encoding, data detection, etc. This scheme provides a reconfigurable HW-based library to implement and simplify the design of communication systems.

Fig. 1 shows the NoC's basic architecture. It comprises a host that handles the NoC management, reconfigurable PEs attached to Network Interfaces (NIs), and the switch component that interconnects the NIs with the host.

This architecture is based on the concept of CoPNoC (Co-Processing Network-on-Chip) [75], with the difference that the network topology used in this work is a star topology, where a central switch will allow parallel data transmission among PEs through a packet-switched method. Therefore, this case study is adequate for the SDR concept and Communication Systems on Chip (ComSoC) [76], [77], [78], [79], [80].

Before discussing the details and challenges of the design and implementation of the switch component, the NoC packet definition, the NoC communication protocol, and the specific requirements for this component are presented.

1) Packet Definition: Fig. 2 shows the packet structure. It is composed of flits of 32 bits. The header stores destination data (address and configuration for the PE) followed by a packet type code used by the NI to process an upper-layer protocol and a packet size field of 9 bits. The packet size field specifies the data size in flits, having a maximum data size of 511 flits. Finally, the tail flit has the source address followed by a cyclic redundancy check (CRC) for packet integrity.

2) NoC Link Signal Protocol: The nodes interconnected in this NoC usually work with their local clocks to comply with communication standards implemented in the SDR system. Therefore, the communication protocol between network components is based on a synchronous version of a two-phase self-timed protocol, where a handshake is performed to synchronize the transmission of flits. Fig. 3 shows two network components communicating through the network channel defined for this NoC and the waveform of a packet transfer. The handshake is accomplished by exchanging the *Req* and *On/Off* signals. Additionally, a *start of packet* (*SoP*) signal is used to detect the header of a packet.

3) Specific Requirements for the Switch: In addition to the generic functional requirements of the described network, to provide flexibility with the number of nodes that the switch will be able to establish communication, the number of switch ports is required to be a parameter for the synthesis process. Furthermore, the arbitration scheme must be enabled for the selection of both round-robin and fixed priority.

4) Design and Implementation Challenges: From the requirements mentioned earlier, the design of a switch component involves an exploration of the design space, ranging from decisions on how to implement flow control, how to process the packet and handle the specific communication protocol, and how to provide the configurability for the number of ports, to finally decide which microarchitecture is the most suitable for this set of requirements and also how to implement it into an HDL.

The design approaches available in the literature could not be suitable for this type of design with specific requirements. For example, approaches based on canonical designs require

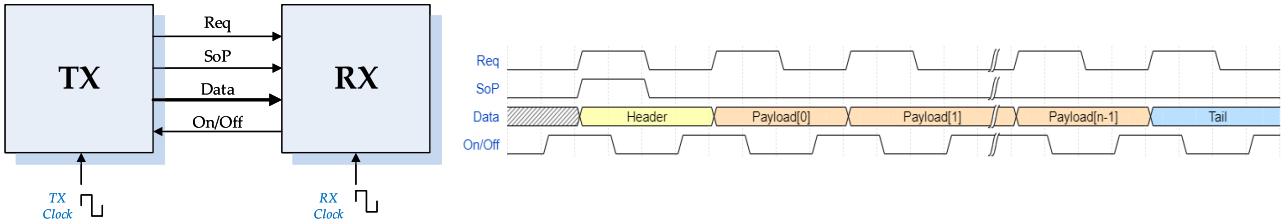


FIGURE 3. Data channel signaling needed to transmit a packet and its waveform. The RX block asserts *On/Off* signal to inform availability to receive a new flit. Then, the TX block places a new flit in the *Data* bus and asserts the *Req* signal to start the handshake. Upon receiving the request, the RX block reads the new flit and resets the *On/Off* signal to indicate that data has been received.

adapting the reference model by modifying, removing, or adding blocks to meet the requirements. On the other hand, the ad hoc designs lack a methodology as they were developed to improve performance. The library-based approaches could be a good option for designing a NoC for a specific application. However, the configurability of its basic building blocks limits the design exploration space. For this reason, a top-down approach is proposed to design and implement the architecture that fits the functional requirements of an application.

III. THE PROPOSED DESIGN METHODOLOGY

This section introduces the concept of abstraction in hardware design for SoCs and how this approach can be utilized in the NoCs domain to design and implement network components such as routers and switches. Firstly, the purpose of describing a system behavior at different levels of abstraction to produce a final implementation is explained. Secondly, the proposed abstraction levels to design NoC components are introduced, and finally, the proposed methodology is described.

A. ABSTRACTION IN HARDWARE DESIGN

Due to the heterogeneity and complexity in the digital design area, a well-accepted approach is based on using one or more models to describe the behavior of a system at a high level of abstraction and then make decisions on its decomposition into hardware and software [81]. From the previous idea, Jantsch and Sander [82] expressed that the design process has to offer a refinement methodology that allows bridging the abstraction gap to yield an efficient implementation.

A typical example of this concept is transaction-level modeling (TLM) [83], which proposes to separate communication details between components from computation details of the components so that the communication is made by transactions (an abstraction of an information transfer). Therefore, when the component to be designed requires communicating with another system component, it will call an interface function of an abstract communication channel. In this way, unnecessary details of the implementation of the communication and computation are hidden, which can be added later. The refinement process in this modeling approach is carried out by three abstraction levels that represent the time accuracy: un-timed, approximate-timed, and cycle-timed. Therefore, a model closer to the final implementation will be obtained if a refinement step is made in the computation or communication processes to get a more realistic timed approximation.

These kinds of methodologies are suited for designs requiring a high-level model that allows designers to evaluate and discuss the system before providing design implementation details. For this reason, the approaches based on modeling at different levels of abstraction are appropriate for designing NoC components that need to meet specific requirements.

B. LEVELS OF ABSTRACTION FOR NETWORKS-ON-CHIP COMPONENTS

NoCs are architectures that provide communication between PEs using packet-switching methods. Therefore, the models generated for a particular network component to be designed should abstract the communication details (interface signaling, protocols for sending and receiving packets) at the higher-level models. Then, new NoC aspects will be integrated through a refinement methodology to obtain a closer approximation to the final implementation. The refinement is based on adding details on how the component should handle packets at each level of abstraction. Fig. 4 shows the proposed levels of abstraction and the relationship between the models generated for designing NoC components.

1) PACKET LEVEL

It is the highest level of abstraction. The component functionality is modeled, abstracting all the communications details through packet-by-packet transfers. Therefore, the buffers of the network component will be allocated in units of packets. This assumption simplifies the description of the component because it is only necessary to describe a black-box system with an algorithm that describes the functionality of a component that receives a packet in a single transaction rather than a complex functional algorithm that processes a packet in flit-by-flit transfers. This level allows the designer to generate a simple black-box model that encapsulates the internal processing of the component designed.

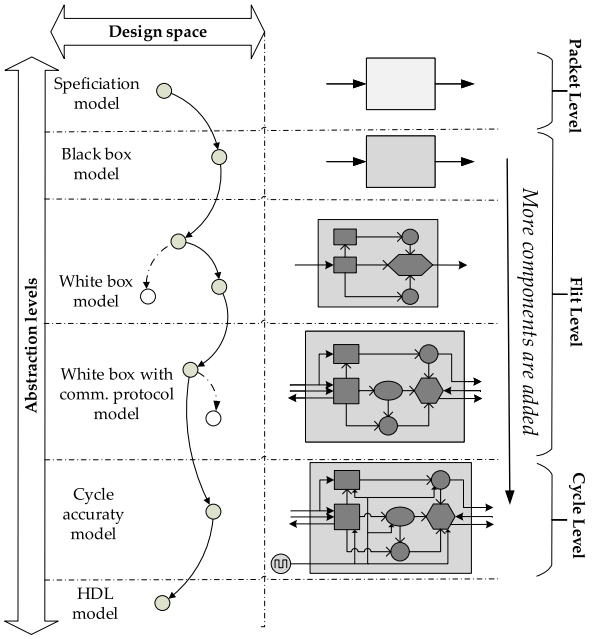


FIGURE 4. Design space exploration for NoC components. The specification model is transformed from a high abstraction level into a implementation model at the lowest abstraction level.

2) FLIT LEVEL

The intermediate level of abstraction. The main feature of this level is that transactions are performed in flit-by-flit transfers. Therefore, this level allows exploring flow control mechanisms. This process involves transforming the packet-level model into a new version with new design aspects that allow a packet to be processed in flit-by-flit transfers. Moreover, new iterations are performed on this abstraction level to transform the black-box model into a white-box model. In this way, the blocks that will make up the microarchitecture and the NoC link signaling protocol will be added systematically.

This level of abstraction aims to get a microarchitecture model that fits the system requirements. Each microarchitecture block will be defined until it can be directly translated into an HDL. That is, a clear definition of a control machine and variables for a datapath definition exist.

3) CYCLE LEVEL

The lowest level of abstraction. The blocks of the latest model generated in the flit abstraction level will be driven by a signal that represents a clock source. In this way, the model behavior will have an execution time like the Register-Transfer Level (RTL).

C. THE PROPOSED METHODOLOGY

The methodology is based on the generation of different models of the component to be designed according to the levels of abstraction shown in the previous section. Hence, starting with a high-level model at the packet level, the subsequent models will be based on the predecessor model, adding new implementation details as allowed by the characteristics of the previous model and the abstraction level. Thus, each model will be spawned around the design space and levels of abstraction, as shown in Fig. 4. Taking this into account, the steps of the methodology are as follows:

1. Definition of System Requirements. The first step is to understand the problem. It is required to know the scope and limitations of the component to be designed to make decisions throughout the design process. Therefore, a set of requirements that specify the design must be defined. Anything out of there will be the designer's choice.

2. Packet Level Model. Once the designer knows the design requirements, a black-box model at the packet level will be described. To simplify the algorithm and understand the basic functionality of the component, the communication between components will be carried on packet-by-packet transfers through transactions described in procedures called *send* and *receive*, as shown in Fig. 5. This model allows the designer to verify the basic component functionality and describe the fundamentals of the algorithm that will be detailed in subsequent models. Furthermore, in this early design stage, the functionality of different routing algorithms can be explored and evaluated based on a metric, if necessary. However, suppose latency and throughput are used as metrics at this abstraction level. In that case, the simulation results must be carefully analyzed since ideal assumptions are made in this model to keep its description simple. These performance evaluations are discussed later in Section V-C3.

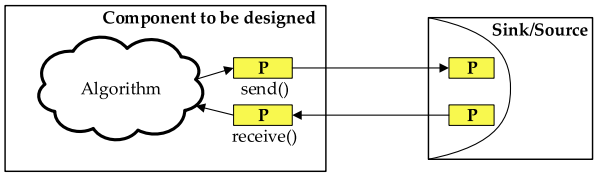


FIGURE 5. Generic packet level model. The main algorithm of the component to be designed receives and sends a packet *P* through the *send* and *receive* procedures.

3. Flit Level Model, Black Box. In this model, an extension of the functionality of the packet-level model algorithm must be added, taking into account that transactions are carried out in flit-by-flit transfers, as shown in Fig. 6. Therefore, the new algorithm must have flow control details to allocate buffer resources and channel usage to each flit of a packet. The *send* and *receive* procedures are redefined to send and receive a packet flit. At this level of abstraction, at least the following models must be generated.

4. Flit Level Model, White Box. Based on the black-box model algorithm at the flit level, an analysis of the decomposition of the basic processing blocks of the algorithm must be performed. The functions defined in the previous model can help to define which blocks will constitute the

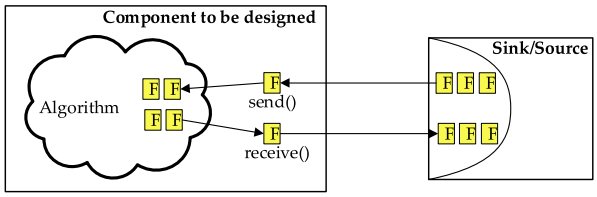


FIGURE 6. Black-box model at flit level. Each packet is sent and received on a flit-by-flit basis.

first approximation of a microarchitecture. Fig. 7 shows an example of the decomposition of an algorithm into blocks. At this point in the design, evaluating flow control alternatives that fit the component requirements is possible. For example, if block B1 in Fig. 7 corresponds to an arbiter functionality, and it is not defined which arbitration scheme must be implemented in the requirements defined in step 1, simulations can be performed, and based on a metric, decide which alternative is better.

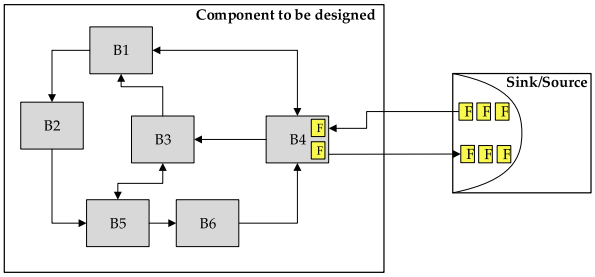


FIGURE 7. Decomposition of the algorithm in processing blocks.

5. Flit Level Model, White Box with Communication Protocol. The signals and processing required for the communication protocol will be added to the new model, as shown in Fig. 8. In this model, the relationship of the communication protocol signals with the blocks of the microarchitecture of the previous model must be described to perform the protocol correctly. An advantage of adding the communication protocol details in this design step is that only the blocks related to the communication protocol will be modified. All other blocks in the microarchitecture will remain the same.

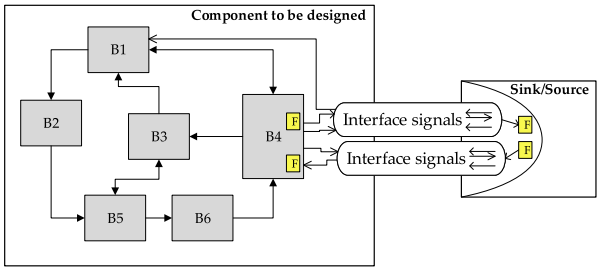


FIGURE 8. Integration of the signals of the NoC communication protocol.

6. Cycle Level Model. In this model, each block of the microarchitecture of the last obtained model will be driven by a signal that will represent a clock source. Also, it must be decided which blocks will not be driven by the clock source, i.e., combinational blocks. Another decision is whether or not a combinational block should have a registered output. Simulations can be performed to analyze the effects of these decisions, which could have repercussions on the latency of the NoC. This level of timed precision allows finding timing issues between blocks of the microarchitecture to correct them before implementing the component.

7. Translation to HDL. In the final step of the methodology, each microarchitecture block will be implemented using an HDL. For this, the control statements and the manipulation of the model variables of each block will be used to define the Finite State Machines (FSMs) and the datapath. Furthermore, in case part of the functionality of any block corresponds to a well-known datapath component, such as memories, arbiters, or crossbars, they can be implemented based on techniques of the literature.

IV. EXPERIMENTAL CASE STUDY: A SWITCH COMPONENT

In this section, the proposed methodology is applied to the design and implementation of the NoC switch explained in Section II. The steps mentioned above in Section III-C will be applied, and the generated models according to the NoC abstraction levels will be discussed.

A. DEFINITION OF SYSTEM REQUIREMENTS

According to Section II, the requirements for the switch component are the following:

- Support for building parameterizable star topologies. The number of switch ports must be defined according to a parameter.
- Selectable arbitration method between round-robin and priority fixed.
- Use of a specialized interface with a synchronous handshaking protocol to connect PEs working with different clock domains.
- Support for the NoC link signal protocol and the packet specification.

These requirements will be used as a starting point to define the packet-level model. Then, based on the characteristics of subsequent abstraction levels and the requirements, decisions will be made to define new implementation details of the switch.

B. PACKET LEVEL MODEL

This model defines which information of the packets will be used to determine the output channel assigned to each incoming packet. Since the switch only supports the star topology, it is not necessary to perform a routing algorithm. Only the direction stored in the header's field *Dest Addr* determines the output channel of the incoming packets.

The models developed with this methodology are described and simulated in the OMNet++ tool [84], a component-based C++ discrete event simulator. In OMNet++, the objects called messages represent events such as the arrival of packets or flits. Also, the messages can represent user-defined control commands to implement delays, timers, processing times, etc., for events that will be executed at a later point in time. This flexibility allows modeling any kind of behavior at different levels of abstraction. Any message arrival to a module is processed by a generic procedure called *handleMessage*, which does nothing by default; thus, the designer must redefine this procedure to add the desired processing algorithm of the incoming messages.

The *handleMessage* procedure for the switch at the packet level is described in algorithm 1. Its input is a message representing an incoming packet to the switch or an incoming control event that indicates that a stored packet has been processed and will depart from the switch. The result of this algorithm is an outgoing packet *pkt_out* that will be sent through its corresponding output port. Another result of this algorithm can be a scheduled control event e that will be delivered to the *handleMessage* procedure itself according to a packet processing time. Messages used in this way are called *self-messages* in OMNet++.

Algorithm 1 checks whether the incoming message is a packet or a scheduled control event. If the received message is a packet (lines 1-8), its destination address is get using the *getDestAddr* procedure, which only takes the value of the packet header's field *Dest Addr*. The output port selected for this packet is computed according to its destination address; furthermore, a modulo operation limits the maximum number of reachable output ports according to the number of ports of the switch. The *getInputPort* procedure gets the corresponding input port where the packet arrived. The *storePacket* procedure stores the incoming packet in the buffer associated to the input port of the received packet. The *processingTime* procedure calculates the packet processing time T as:

$$T = t_r + L/b, \tag{1}$$

where t_r is the header processing latency in cycles, L is the packet length in flits, and b is the link bandwidth. The term L/b determines the serialization latency. We assume that a header requires two clock cycles to be processed, and the ideal bandwidth of the links is one flit per clock cycle without making a handshake. This consideration is for keeping the model as simple as possible. Finally, each switch queue will have a capacity for a single packet because the data is processed with packet granularity.

The *schedulePacket* procedure grants access packets to their desired output port. Furthermore, this procedure generates a *self-message* e that will be scheduled according to the packet processing time T of the packets. If the message is a control event (lines 10-12), the information about the packet that has been processed is obtained via the *getEventInfo* procedure. Then the stored packet is removed from the buffer using the *popPacket* procedure, and finally, it is sent to its corresponding output port through the *sendPacket* procedure.

Algorithm 1 HandleMessage Procedure at the Packet Level

Input: Message *msg*, that can be an incoming packet or an event for sending a stored packet.
Output: Outgoing packet *pkt_out* or a scheduled event *e*.

```
1:  if msg is a packet then
2:      pkt = msg
3:      dest_addr = getDestAddr(pkt)
4:      output_port = dest_addr mod num_ports
5:      input_port = getInputPort(pkt)
6:      storePacket(pkt, input_port)
7:      time = procTime(pkt)
8:      e = schedulePacket(output_port, input_port, time)
9:  else
10:     [output_port, input_port] = getEventInfo(msg)
11:     pkt_out = popPacket(input_port)
12:     sendPacket(pkt_out, output_port)
13: end if
```

This model provides a first idea of what the switch will need for its implementation. For example, line 4 of algorithm 1 shows a high-level description of how the output channel should be computed. For this component, it is directly calculated. However, if the component to be designed is a router, a routing algorithm could be defined and tested by simulation.

C. FLIT LEVEL MODEL: BLACK BOX

At this level of abstraction, the packet is processed with a granularity of flits. Therefore, the black-box model has to manage the buffers and channels in units of flits. This model can explore different flow controls that allocate resources in units of flits; the most common are wormhole and VC. Wormhole flow control was chosen over VC because the packet header specified for this NoC lacks a specified field to handle different information flows. Also, wormhole flow control is simple and allows the switch to have a reduced amount of storage.

In order to use wormhole flow control, each input port has an associated control state called *channelState* to track the status of the current packet being processed. The finite state machine (FSM) related to each input port is shown in Fig. 9. The FSM will remain in the IDLE state when no packet is processed. Access to an output port will be requested if a new packet is received, and the FSM will change to the WAIT_GRANT state. Then, the FSM will remain in this state until the output port has been granted. Subsequently, the FSM moves to the ACTIVE state, and after the packet has been sent, the FSM will return to the IDLE state.

The *handleMessage* procedure for the switch at the flit level is described in algorithm 2. Each incoming flit is stored according to its arrival input port (lines 1-4). If the input channel state is IDLE and the incoming flit is a header,

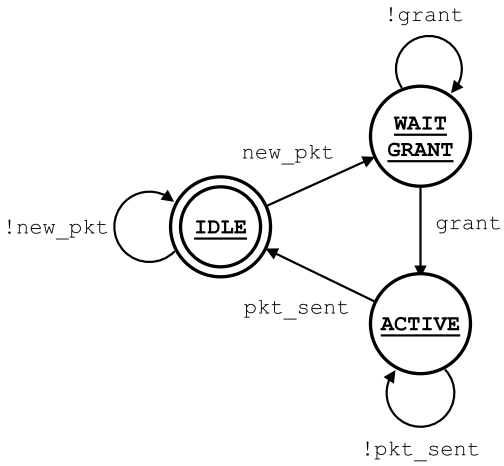


FIGURE 9. FSM associated with each input port of the switch. This FSM is used to track the current status of each packet being processed in the switch.

a request will be performed to try scheduling the new flit to its requested output channel through the *scheduleFlit* procedure. This procedure returns a scheduled event *e* for sending the stored flit and a variable called *grant* that will be used to assign a new state to the corresponding *channelState* using the *processChannelState* procedure (lines 5-9).

The *scheduleFlit* procedure is similar to the *schedulePacket* procedure. It creates an event (*self-message*) to send a flit, but now it considers each input port state. If an output port is already assigned to a packet of an input channel, the other input channels will have to wait until the output channel is released. Therefore the waiting input ports will have assigned the WAIT_GRANT state.The *checkWaitingChannels* procedure is used to allocate a recent output channel released to a new packet stored on an input port waiting for a grant.

The *handleMessage* procedure of this model has a similar structure to the packet level model (an *if* statement to store incoming packets and an *else* statement to process the event for sending a flit). Notice that the newly added procedures provide the necessary processing to handle the flow control selected. The *storeFlit*, *popFlit*, and *sendFlit* procedures have the same functionality as their equivalents of the packet level model, but now the messages are handled like flits. The output port computation remains unchanged.

The flit transfer time from the switch to its destination will be specified using the *scheduleFlit* procedure. This delay is set to two clock cycles, considering ideally that one clock cycle is used to traverse the switch datapath and another for link traversal. Furthermore, it will be considered that the header flit adds an extra clock cycle for requesting its desired output port. In order to get a first and rough approximation to the link signal protocol to provide buffer backpressure, an *On/Off* variable is used to determine when the downstream node is ready to receive the next flit from the switch. If the *On/Off* variable is enabled, a new flit will be transferred, and the *On/Off* variable will be disabled. When the downstream node receives the incoming flit, it will enable the *On/Off* variable to inform availability to receive the next flit. This behavior would be an abstraction of the handshake that will be detailed in later models.

Algorithm 2 HandleMessage Procedure at the Flit Level, Black Box

Input: Message *msg*, that can be an incoming flit or an event for sending a stored flit.
Output: Outgoing flit *flit_out* or a scheduled event *e*.

```
1:  if msg is a flit then
2:      f = msg
3:      input_port = getInputPort(f)
4:      storeFlit(f, input_port)
5:      if f is header and channelState[input_port] = IDLE
    then
6:          dest_addr = getDestAddr(f)
7:          output_port = dest_addr mod num_ports
8:          [e, grant] = scheduleFlit(output_port, input_port)
9:          processChannelState(input_port, grant)
10:     end if
11: else
12:     [output_port, input_port] = getEventInfo(e)
13:     flit_out = popFlit(input_port)
14:     sendFlit(flit_out, output_port)
15:     if channelState[input_port] = IDLE then
16:         [e, grant] = scheduleFlit(output_port, input_port)
17:         processChannelState(input_port, grant)
18:     else
19:         e = checkWaitingChannels(output_port)
20:     end if
21: end if
```

D. FLIT LEVEL MODEL: WHITE BOX

The blocks that make up the white-box model are defined based on the procedures defined in the black-box model described above. The algorithm of the *handleMessage* procedure of algorithm 2 provides a structure on how the switch should process the flits. Firstly, each incoming flit is stored in a buffer. Then, a channel state is processed according to three elements: the arrival port of the flit, the message (that can be either a flit or event to send a flit), and the result of the *scheduleFlit* procedure. Therefore, the flit buffering and the algorithm of the *processState* procedure can be integrated into a block to process received flits. The kind of behavior of this block is similar to an input port of a generic router.

Secondly, since the *processState* procedure needs to know when it can send a flit, it is in communication with the *scheduleFlit* procedure, so this function corresponds to the behavior of an arbiter that grants an output port only to an input port. Also, the *scheduleFlit* procedure generates the event to send a flit stored and enables the input port to update its channel state. The event to send a flit is an abstraction of the interconnection of an input and an output port that processes the *sendFlit* procedure. Therefore, the arbiter block should control an interconnection block that will be called the

crossbar. The *checkWaitingChannels* procedure is integrated into the arbiter block since it only checks which input ports have requested the released output port. Next, it calls the *scheduleFlit* procedure to decide on a new granted input port. Notice that the arbiter can be implemented with round-robin and the priority fixed schemes required. Fig. 10 shows the main blocks of the white-box model and the relationship with the procedures of the black-box model.

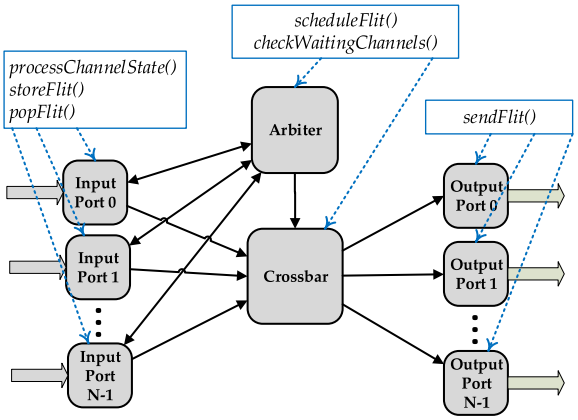


FIGURE 10. Main modules of the white-box model at flit level and their relationship with the black-box model functions.

The communication among blocks of the white-box model is made through messages. The block `Input Port` sends messages representing requests to the `Arbiter` block and receives a message when a grant is generated to its requested output port. Also, the `Arbiter` block sends a control message to configure the `Crossbar` block according to the granted ports. The `Crossbar` block only forwards incoming flits to a specific `Output Port` block.

The model of each block is described according to the algorithm of the procedures they represent. Furthermore, if necessary, the modules' control can be implemented using FSMs. OMNet++ provides a class and macros to build and debug FSMs. For example, for the `Input Port` block, an FSM was designed according to the states and behavior defined in the black-box model.

The white-box model has the same functionality and flit delay as the black-box model. The only difference is the white-box model shows the internal block architecture. Thus, the designer establishes how the flits and control events interact between blocks without internal timing details. For this reason, the flit delay is established as the same as defined in the black-box model, and the `Arbiter` block computes it when a new flit is authorized to depart from the switch.

E. FLIT LEVEL MODEL: WHITE BOX WITH COMMUNICATION PROTOCOL

At this point of the design, the implementation details required for the communication protocol are added. Each signal of the protocol is added to the model, and it is defined which blocks of the microarchitecture of the previous model will drive these signals and how they will be used to send and receive flits. According to Section II, the protocol communication uses an *On/Off* flow control to indicate when the receiver is ready to receive a new flit, and then the handshake is performed. Therefore, the `Input Port` and `Output Port` blocks will perform this handshake. On the other hand, the `Arbiter` block will require to know the availability to send a new flit of each `Output Port` block to generate grant signals to the `Input Port` blocks that generate requests. Hence, adding a status signal from each `Output Port` to the `Arbiter` block will be necessary. Fig. 11 shows the modifications added to the microarchitecture of the previous model.

According to the buffer occupancy and the NoC link signal protocol, the `Input Port` block drives the *On/Off* output signal. If the buffer is not full, this block will perform a handshake to receive a new flit. The *SoP* and *Req* signals are used to perform the protocol and notify when the *Data* input signal is valid. Similarly, the `Output Port` block will be able to send a new flit when the input signal *On/Off* is enabled and will notify the `Arbiter` block (blue dotted line in Fig. 11) of this availability.

In this model, the link traversal time for a flit is provided by a functional approximation of the handshake delay performed using the protocol signals. When the handshake is carried out, one clock cycle is considered between each signaling event. The flit traversal delay within the switch is one clock cycle, as the predecessor model with the extra cycle for flit header processing.

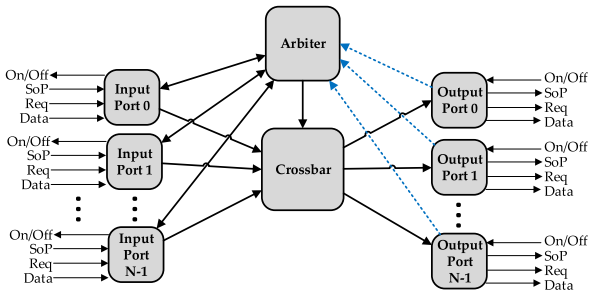


FIGURE 11. Modifications to the white-box model when the communication protocol is integrated. New internal connections (blue dotted lines) are added to inform the `Arbiter` of the status of the handshake protocol.

F. CYCLE LEVEL MODEL

A module that acts as a clock source is generated to add a cycle-timed precision to the last model at the flit level, which periodically sends an event that will trigger the *handleMessage* procedures of the microarchitecture blocks. The `Input Port` and `Output Port` use state machines to perform their processing and need variables to store temporal values. The `Arbiter` block also uses internal variables to track the priorities assigned in each arbitration

round. Therefore, according to its internal processing, all blocks but `Crossbar` were selected to be sequential for this microarchitecture. The `Crossbar` block could have registered outputs, but the drawback would be to have an extra clock cycle in the switch traversal of each flit.

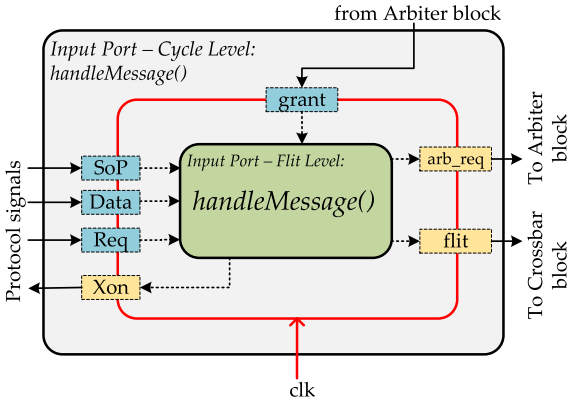


FIGURE 12. Cycle level model of the `Input Port` block. All received signals from the extern modules will be processed by the *handleMessage* procedure of the `Input Port` block at the flit level in each clock cycle.

The basic structure of the blocks at the cycle level is based on using the latest *handleMessage* procedure of each block at the flit level. However, these procedures will be triggered when a clock cycle event is received. Each block will store the incoming/outgoing messages from/to other blocks to achieve this behavior. For example, Fig. 12 shows the structure of the `Input Port` block at the cycle level. The *handleMessage* procedure at the flit level from the `Input Port` block will process all incoming messages and generate the corresponding outgoing messages. However, these messages will be registered in variables in each clock cycle. Therefore, with these input/output registers, the structure of a generic sequential circuit (processing logic with input and output registers) is added to the model. For the synchronization of the handshake control signals, an extra delay of one cycle was added to the *Req* and *Xon* signals to model the behavior of a signal crossing a clock domain using a standard two flip-flop synchronizer [85].

OMNet++ supports object-oriented programming and the use of polymorphism. Therefore, it allows the cycle-level model to inherit the procedures from the latest model at the flit level. Then, the *handleMessage* procedure of the model at the clock cycle level will be able to use the *handleMessage* procedure model at the flit level, as is shown in Fig. 12.

G. TRANSLATION TO HDL

The `Input Port`, `Output Port`, `Crossbar`, and `Arbiter` models are translated to an HDL. Verilog standard was used for this case study. The HDL code is obtained easily (without using an automatic tool in this paper) because the details of each block microarchitecture have already been defined throughout the design process; input/output signals, control statements, and variables manipulation to perform its processing. Fig. 13 shows the microarchitecture obtained according to the switch cycle-level model, which only shows an `Input Port` and an `Output Port`. The control unit of the `Input Port` and `Output Port` blocks is already detailed in their FSMs from the OMNet++ models. Since the bitwidth of the control signals between modules grows according to the number of ports, encoders and decoders are used to reduce the bitwidth. Also, a FIFO buffer is needed, which in the literature already exists efficient implementations. The `Crossbar` was implemented using multiplexers, and the `Arbiter` block was built using variable priority iterative arbiters [6], one for each `Output Port` of the switch. Furthermore, a block was added that manages round-robin priorities associated with each `Input Port`.

V. RESULTS

In this section, the proposed methodology is analyzed and compared in qualitative and quantitative terms. First, comparing the generalities of the literature approaches with the proposed approach is made to identify similarities and differences. Later, the design and implementation of the case study are analyzed in terms of latency, throughput, and area.

A. DESIGN APPROACHES COMPARISON

Many authors have designed different router architectures to improve a metric of interest. The findings of this work suggest that there exist three router design approaches explained in Section I. Each approach implies certain particularities and methods to design a router microarchitecture and, if possible, its implementation. Therefore, four characteristics involving router design are proposed to make a qualitative comparison between design approaches. (1) Strategy for architectural exploration. Each methodology has a procedure to decide which NoC design aspects will be addressed. (2) A defined design flow. A design flow is a sequence of steps or processes to perform the design cycle. (3) The use of frameworks or tools to automate the design flow. (4) Generation of the design implementation. Some works only focus on designs with cycle-level accuracy without hardware implementation.

Table 2 shows a comparison between the design approaches of the literature and the proposed methodology. Approaches based on canonical architectures allow selecting a specialized router design aspect, such as buffer organization, low power, pipeline, datapath, or link design, which will be the design starting point. On the other hand, the library-based approaches allow choosing and configuring a predetermined set of features covering different router design aspects. A well-known case is the *xpipes* library [74], which allows selecting the number of ports, the number of VCs, and buffer size. Therefore, regarding to architectural exploration, these two approaches are limited to a specific design aspect and the level of configurability of the building blocks. Furthermore, both have a defined design flow, and some works have a hardware implementation. Also, a great

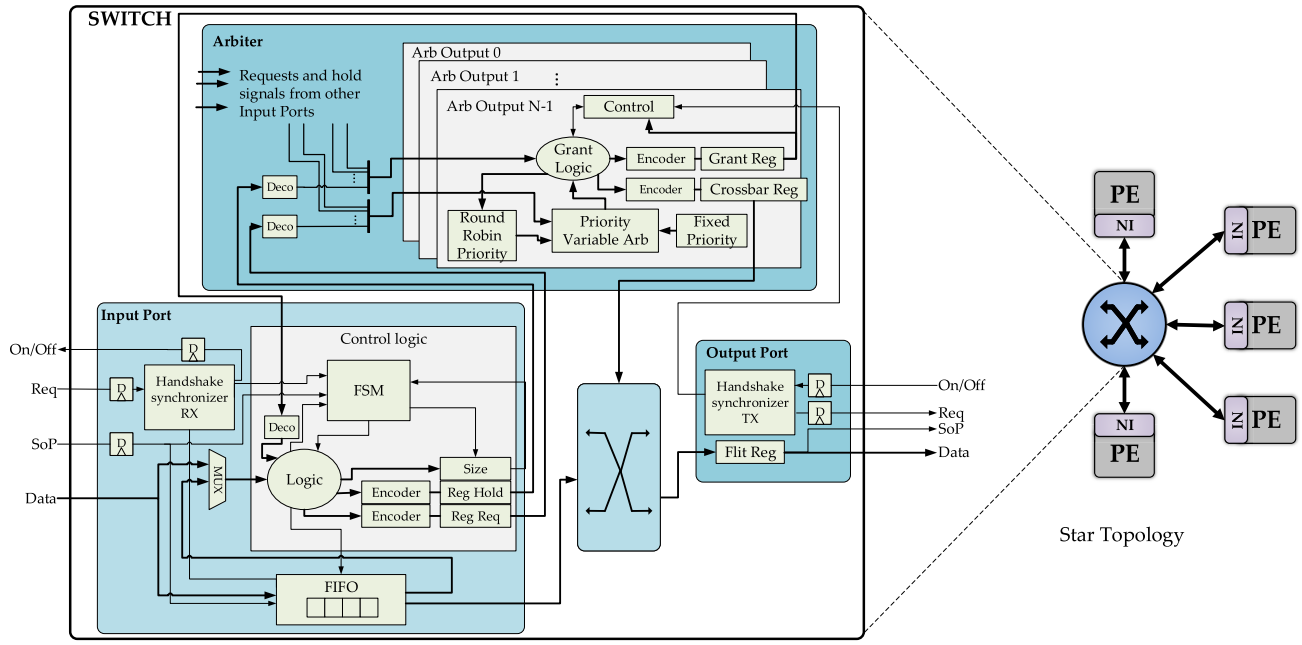


FIGURE 13. The switch microarchitecture obtained by using the proposed methodology.

TABLE 2. Router design approaches generalities.

Approach	Architectural Exploration	Design Flow	Automation	Hardware Implementation
Based on Canonical Architectures	Limited to a specialized area	Yes	No	Some works
Library-based	Limited to configurability of the basic building blocks	Yes	Yes	Yes
Ad hoc	According to requirements	No	No	Some works
The proposed methodology	According to requirements	Yes	No	Yes

advantage of library-based approaches is the possibility of generating tools that automate their design flow because they can adopt algorithmic strategies to select the optimal configuration values for their building blocks.

Alternatively, ad hoc approaches are free to explore any design alternatives, and as a result, the architectures designed are novel to improve NoC performance. Nevertheless, this flexibility in exploring the design space comes at the expense of an undefined design flow and relies on designer's vision and understanding.

It should be emphasized that the design choices made in the above approaches are evaluated with cycle-accurate results because they are focused on improving performance. In contrast, the proposed methodology considers the need to explore the design space without limiting it to a specialized design area or a set of configurable building blocks. The freedom of exploration will be according to the functional system requirements. Therefore, this exploration approach differs from the state-of-the-art because the design choices are taken to functionally meet the specified requirements according to the processing granularity of each abstraction level rather than being made to improve performance.

However, suppose a determined functionality (flow control, routing algorithm, arbitration, etc.) cannot be associated with a requirement, and the designer has the freedom to select it. In that case, evaluations can be performed at the corresponding level of abstraction based on a metric to decide which alternative is better. Nevertheless, a careful performance analysis must be made at high levels because ideal assumptions are made. This performance analysis is discussed later in Section V-C3.

Additionally, a top-down approach is employed, which allows defining a design flow to establish the steps to carry out the design and implementation of the required component. It is noteworthy that this kind of approach focused on the NoC design according to the system requirements, with a flexible exploration design space, is similar to generic SoC design approaches. However, the proposed work adapted it to be NoC oriented through the proposed abstraction levels to model the component to be designed. The abstraction levels are oriented to the characteristics of the NoC communication, providing a structured approach to tackle design aspects without thinking about the implementation details in the early stages.

B. PERFORMANCE RESULTS

In this section, the switch performance under different synthetic traffic patterns and traces of real workloads is evaluated. Simulations are performed using the cycle-level model described in Section IV-F. The NoC topology is a star (as is shown in Fig. 13), and the NoC parameters are the number of ports of the switch, arbitration type, size of the input buffers, packet size, and the operation frequency of the switch (ClkSW) and terminals (ClkT). The terminals replace the PEs and act as a sink and source for the NoC packets for statistics recollection.

1) SYNTHETIC TRAFFIC

Three network traffic patterns [6] were used: (1) uniform random, where each terminal is equally likely to send packets to each destination, (2) bit-complement, a permutation where each source terminal sends traffic only to a single destination, the destination is computed complementing each bit of the address of the source terminal and (3) hot-spot, where the probability of sending a packet to the hot-spot terminal is greater than the probability of sending a packet to the other terminals, for this experiment, the hot-spot terminal corresponds to the terminal with address zero, and its probability is 0.4. The remaining traffic is distributed uniformly among the other terminals.

The performance impact of the switch is explored with the above three traffic patterns, using a different number of ports and different operating frequencies according to the configurations shown in Table 3. The ClkSW frequency value is 50 MHz because it is the frequency available by the oscillator of the Altera field-programmable gate array (FPGA) device where the switch was synthesized and tested. The different ClkT settings are used to know the performance that can be achieved using frequencies slower or faster than the switch.

The metrics used to evaluate the performance are the average latency of the received flits and the throughput in flits/cycle, both in terms of the operating frequency ClkT. In order to maintain homogeneity in the initialization of simulations and considering that each frequency configuration will determine different rates at which packets are received, a warming period of 10,000 cycles was used. This allows to reach a steady-state for all configurations. A measurement phase of 50,000 cycles was chosen to collect at least five times more samples in each run than those discarded in the warm-up phase. For statistic soundness, 15 repetitions were made with different random seeds.

Fig. 14 shows the results of these experiments. According to the latency results (Figs. 14a to 14c), it is observed that the permutation traffic has the best saturation throughputs[2] because there is never contention between packets; all switch resources are allocated to each source-destination pair without conflict. On the other hand, hot-spot traffic shows the worst saturation throughputs. This behavior is expected because slightly less than half of the traffic is sent to a single destination. So there will be a higher number of contentions for that output port because there is only one path to reach it.

TABLE 3. NoC configurations for the simulations using synthetic traffic.

Parameter	Description
Operation frequencies	ClkT = 50 MHz, ClkSW = 50 MHz ClkT = 20 MHz, ClkSW = 50 MHz ClkT = 100 MHz, ClkSW = 50 MHz
Number of ports	4, 8, 16
Arbitration	Round-Robin
Input buffer size	4 flits
Packet size	5 flits

The performance under uniform traffic is between that obtained with permutation and hot-spot traffic. Although performance under uniform traffic does not have high saturation throughputs like permutation, its performance does not drop considerably as hot-spot traffic when the number of switch ports increases beyond four ports.

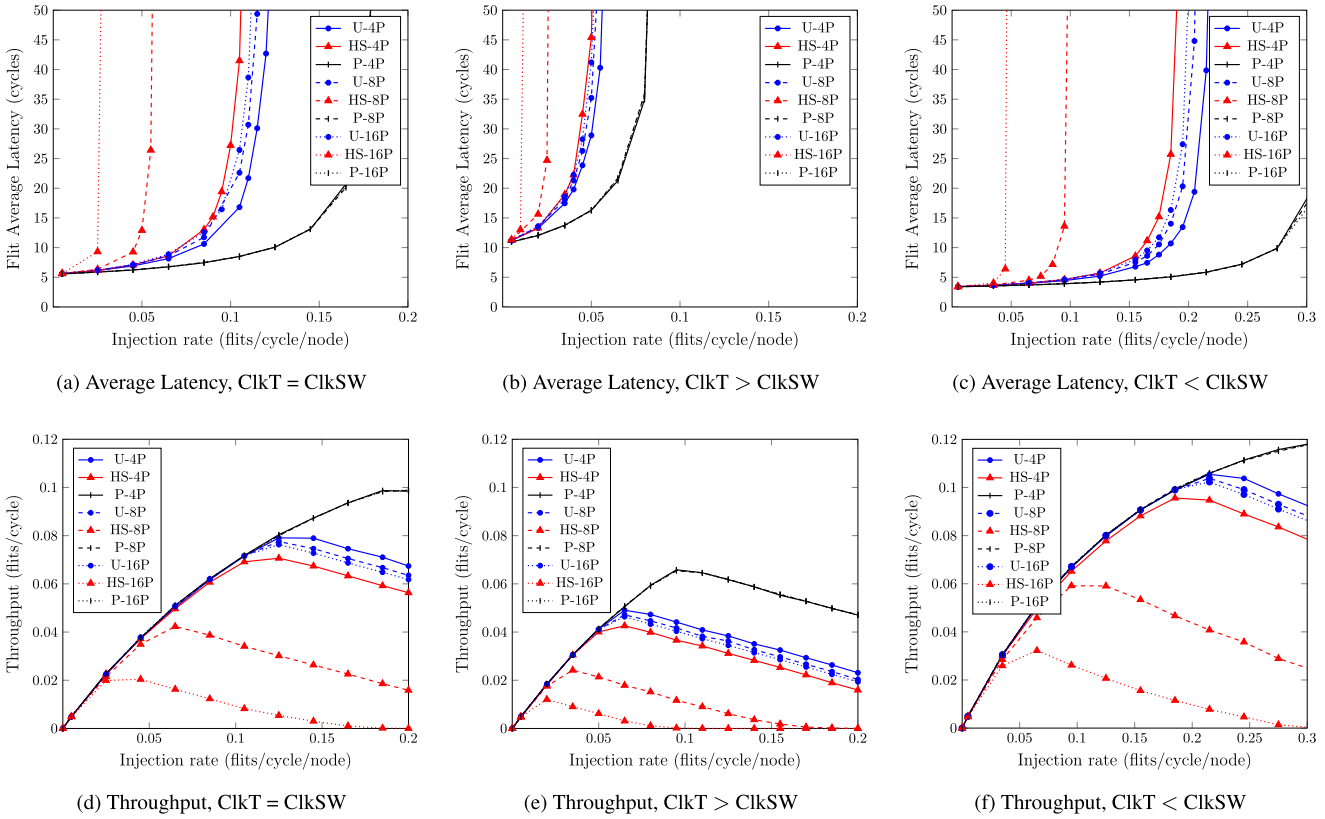
Regarding the results using different operating frequencies for ClkT, the reported curves have similar behavior with different saturation throughputs according to the relationship between ClkT and ClkSW. Fig. 15a shows the saturation throughput in bits per clock cycle of the terminal for each experiment according to the number of the ports used (4P, 8P, and 16P) and the different operating frequencies used: case 1 (C1) where ClkT = ClkSW, case 2 (C2) where ClkT > ClkSW and case 3 (C3) where ClkT < ClkSW.

Although it seems case C3 has the best saturation throughput and case C2 the worst, it must take into account that ClkT in case C3 has a frequency of about half of the ClkSW, and ClkT frequency is double the ClkSW in case C2. Therefore, using the same ratio between ClkT and ClkSW described above, different saturation throughputs in Mbps can be obtained according to the clock frequencies used. For example, Fig. 15b shows the saturation throughputs in Mbps using the values of the clock frequencies reported in Table 3.

It can be noticed that traffic patterns such as permutation with a 4, 8, and 16-port switch, traffic of up 300 Mbps can be injected before the NoC is saturated; on the other hand, the worst performance is under hot-spot traffic with a 16-port switch with saturation throughputs between 30 and 50 Mbps. Also, the saturation throughput of case C3 in all simulations is approximately 30% lower than cases C1 and C2, which have similar saturation values. This performance degradation is because the ClkT is slower than ClkSW. Therefore, the time elapsed between data flit transfers through the handshake is less than ClkT frequencies equal to or greater than ClkSW.

The accepted traffic curves (Figs. 14d to 14f) have similar behavior in all experiments. It can also be observed that the throughput drops drastically after saturation under hot-spot traffic patterns with an 8, and 16-port switch. This behavior under hot-spot traffic is expected since each terminal sends

[2]Saturation throughput is defined as the injection rate when the network is saturated. Generally, it is measured as the injection rate when the average latency is three times the zero-load latency [17], [86].



(a) Average Latency, ClkT = ClkSW (b) Average Latency, ClkT > ClkSW (c) Average Latency, ClkT < ClkSW

(d) Throughput, ClkT = ClkSW (e) Throughput, ClkT > ClkSW (f) Throughput, ClkT < ClkSW

FIGURE 14. Latency and throughput curves of the switch using uniform (U), hot-spot (HS) and permutation (P) synthetic traffic patterns, the number of ports considered are 4 (4P),8 (8P) and 16 (16P).

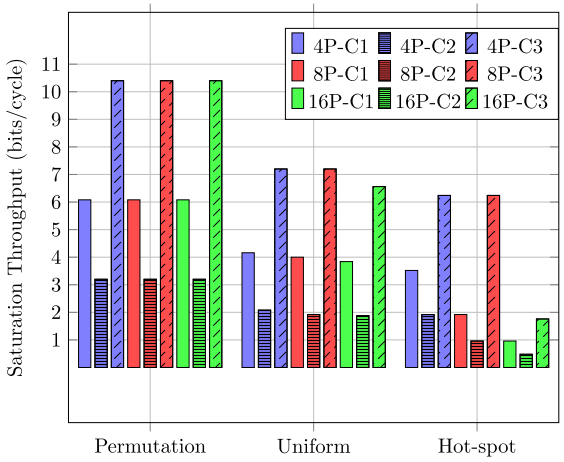
40% of its traffic to only one destination. Consequently, in the worst case, when all terminals send a packet to the hot-spot destination, they will have to wait at most n arbitration rounds (due to the round-robin scheme) to be granted the desired output port, where n is the number of ports of the switch.

Fig. 16 shows how the latency and throughput curves of the switch using round-robin (RR) and the switch using the priority-fixed (PF) scheme compare. Results for the 8-port and 16-port switches are presented because significant differences were noted under hot-spot traffic. Figs. 16c and 16d show that the accepted traffic under the hot-spot pattern with PF does not drop drastically compared to RR. This result is because the terminal with priority will always send more packets to the hot-spot terminal without interruption. Thus, this terminal will receive more packets on average at the cost of generating starvation with the other terminals.

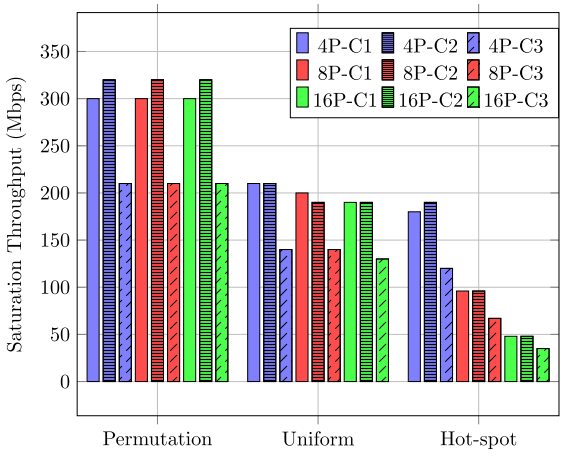
In addition to the above results, a simulation to evaluate the NoC performance using different packet sizes in a NoC with eight terminals is presented. Table 4 shows the packet sizes the terminals send to their destinations in this scenario. The simulation results are presented in Fig. 17; a comparison is provided against the previous experiment when the packet length is fixed. The zero-load latency with uniform and hot-spot traffic is five cycles greater on average than the obtained under permutation traffic. Also, under hot-spot and uniform traffics, the average latency increases asymptotically at low injection rates. This result shows that using large packet sizes degrades the performance of this NoC using a switch component. This behavior is because the larger the packet sent by a terminal, the longer the switch resources will be used to send a packet. Hence, the waiting time of the other terminals that want to send a packet to the same destination will increase.

The accepted traffic curves show an interesting finding for hot-spot traffic. With major injection rates than the saturation throughput using hot-spot traffic (0.7 flits/cycle/node), the network throughput of the experiment that uses fixed-length packets is 30% less compared to the experiment that uses variable-length packets. However, these results must be interpreted with caution because the throughput reported is the average per node. Therefore, a more careful analysis based on reporting the minimum throughput across all network flows (source-destination pairs) is shown in Fig 18.

As can be observed, in the experiments using variable-length packets, the minimum throughput drops drastically to zero after reaching saturation, revealing a fairness problem. Furthermore, the simulations reported that the minimum throughput is from the flows that generate the smallest packets (the first two terminals), showing that



(a) Saturation throughput in terms of the clock cycle of the terminals.



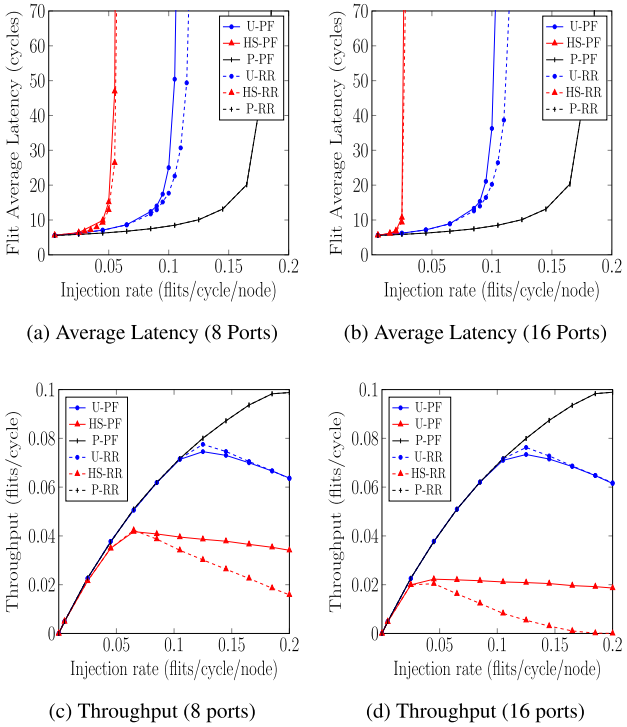
(b) Saturation throughput in Mbps according to the operating frequency values reported in Table 3.

FIGURE 15. Saturation throughputs for different synthetic traffic patterns. The number of ports considered are 4 (4P),8 (8P) and 16 (16P). Three different cases are considered: ClkT = ClkSW (C1), ClkT > ClkSW (C2), and ClkT < ClkSW (C3).

these flows are throttled after saturation despite using the round-robin arbitration scheme. This throttling is because to the round-robin scheme provides an equal service regarding the number of times one requester is served. However, when using variable-length packets as in the above experiment, terminals that send packets with the largest size will send more flits than terminals that send packets with the smallest sizes. In consequence, a weighted round-robin scheme should improve the fairness problem.

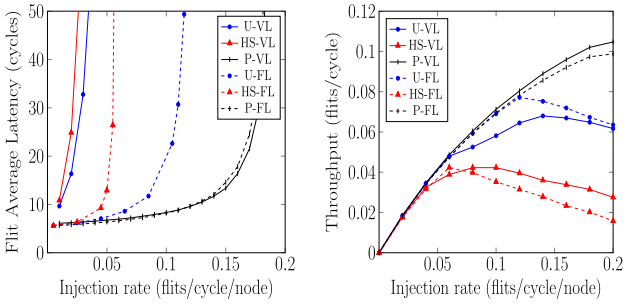
TABLE 4. Packet sizes configured for terminals.

Terminal address	Packet size
0,1	5 flits
2,3	63 flits
4,5	100 flits
6,7	210 flits



(a) Average Latency (8 Ports)
(b) Average Latency (16 Ports)
(c) Throughput (8 ports)
(d) Throughput (16 ports)

FIGURE 16. Latency and throughput curves of the switch using uniform (U), hot-spot (HS) and permutation (P) synthetic traffic patterns with round-robin (RR) and priority-fixed (PF) schemes.



(a) Average Latency
(b) Throughput

FIGURE 17. Latency and throughput curves of the switch using uniform (U), hot-spot (HS) and permutation (P) synthetic traffic patterns. Comparison between experiment using packets of variable-length (VL) against experiment using fixed-length (FL) packets.

2) TRACES OF REAL WORKLOADS

The NoC performance was evaluated using real traffic traces from the MCSL suite [87], which provides traffic information of different real applications optimized for regular NoC architectures such as torus, mesh, and fattree. The MCSL suite includes multiprocessor System-on-Chip (MPSoC) applications, which are used in heterogeneous architectures. For this reason, these benchmark applications are adequate to evaluate the proposed work.

The authors of the MCSL suite model each real application using Task Communication Graphs (TCG) to provide communication dependencies between the tasks of an application. Then, algorithms are applied to get task mapping and

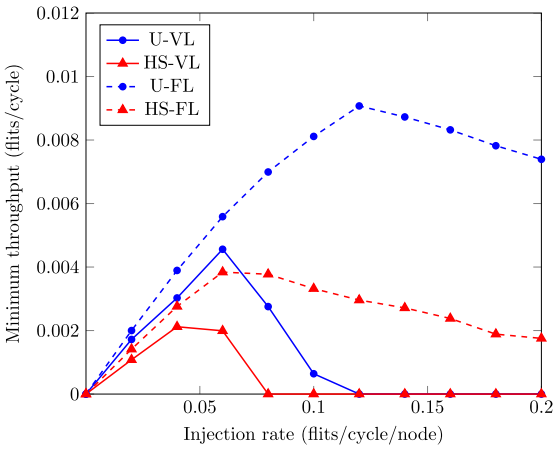


FIGURE 18. Minimum throughput across all flows of the switch using uniform (U) and hot-spot (HS) synthetic traffic patterns. Comparison between experiment using packets of variable-length (VL) against experiment using fixed-length (FL) packets.

schedule for a target NoC topology according to each task's dependencies and processing times. Finally, they simulate the application optimized for the NoC topology and generate a file with the recorded traffic pattern.

Another aspect of the traffic patterns is that the packets injected into the NoC by the processing blocks (PBs) will depend on each task processing time, schedule, and dependencies on other tasks. Therefore, each task will not be executed in its assigned processing block until it has received the data required to start its execution. Then, when each task finishes its execution, it will send its processed data to the tasks that depend on it.

The MCSL suite provides traffic patterns for applications such as a Fast Fourier Transform (FFT) of 1024 samples, Reed-Solomon (RS) coder and decoder, sparse matrix solver, etc. Each of them has a specific number of tasks and communication links according to its modeled TCG. Each link indicates that a packet will be sent from a source task to a destination task. Table 5 shows the characteristics of the applications taken from the MCSL suite used for the simulations of this work and the statistics of the packet lengths that the tasks of each application produce.

TABLE 5. Summary of the applications modeled in the MCSL suite and their packet lengths statistics.

App General Information			Packet Length Statistics (flits)		
Application	No. tasks	No. links	Avg. Size	Max. Size	Min. Size
Robot	88	131	51	64	28
SPARSE	96	77	204	256	124
RS-32-28-8 dec	182	392	3	3	3
RS-32-28-8 enc	262	348	3	3	3
FPPPP	334	1145	54	66	28
FFT-1024	16384	25600	5.6	6	4

The applications have a variety in terms of the number of tasks and communication links. Applications like the RS encoder and decoder only send small, fixed-size packets (3 flits). SPARSE sends a small number of packets, but the average packet size is 204 flits. On the other hand, the FFT is the most demanding application because it has the largest number of packets to send to the NoC.

The recorded traffic patterns optimized for a fattree topology were used because, among the topologies available in the suite, it is the most similar to the star topology studied in this work. Traffic patterns for 4, 8, and 16 PB were used. The switch configurations are round-robin arbitration and 8-flit buffer size.

Fig. 19 shows the execution times of each application normalized to the base case (4-port switch). It can be observed that in most applications, as the number of ports increases, the execution time reduces. The applications ROBOT, SPARSE, and FPPPP have an average reduction of 30% in execution time using an 8-port switch and a reduction of up to 50% using a 16-port switch. The FFT has a reduction of 8% and 20% for a 4-port and 8-port switch, respectively. On the other hand, the RS encoder execution time grows 18% and 30% using 8-port and 16-port switches, respectively, despite having a medium number of tasks and communication links compared to the other applications.

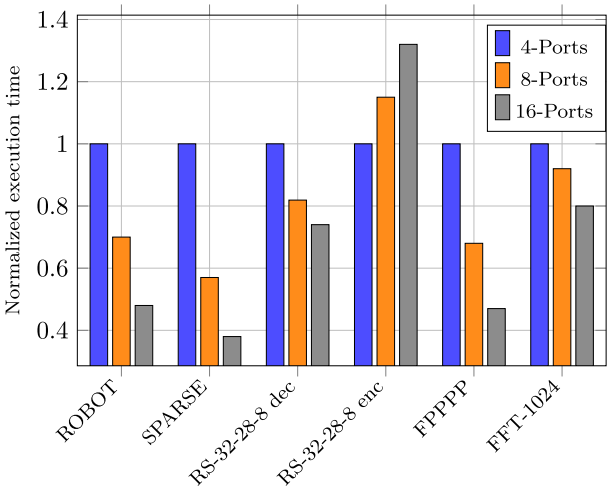


FIGURE 19. Execution times normalized to the case base (4-Ports switch).

In all applications but the RS encoder, the execution time is reduced. This result makes sense because the greater the number of PB available for an application, the greater the distribution of the tasks to take advantage of parallelism. To analyze the results and why the execution time increments for the RS encoder when the number of PB increases, the spatial distribution of the packets for each application was obtained, counting the number of packets delivered in each output port of the switch.

Most applications distribute the generated packets among all PBs, but there is a concentrated load on a few switch ports

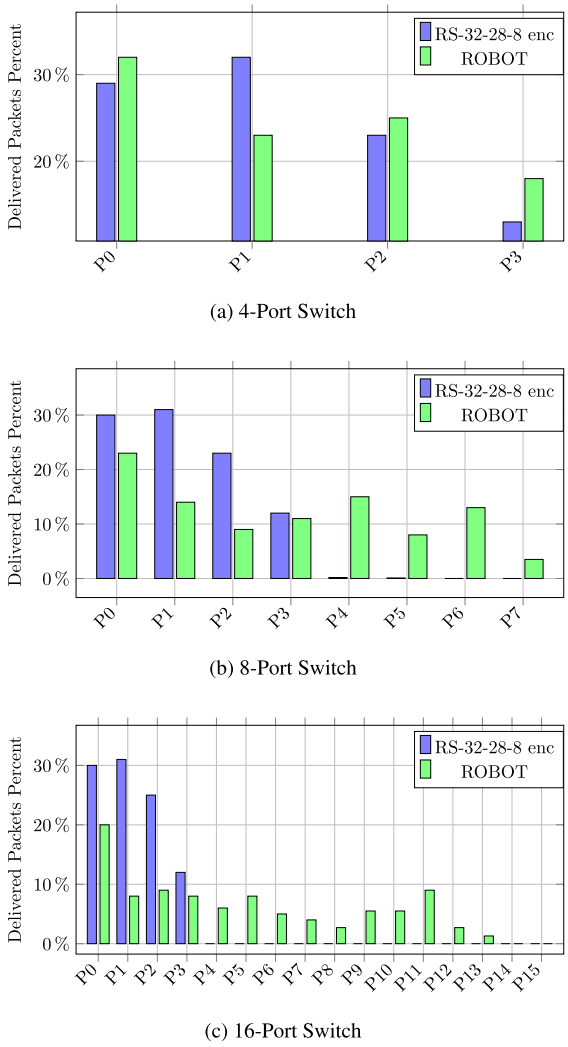


FIGURE 20. Amount of delivered packets on each switch output port using RS encoder and ROBOT applications.

in the packet distributions for the RS encoder. Fig. 20 shows the packet distributions for the RS encoder and SPARSE applications to compare configurations with 4, 8, and 16-port switches. The packet distributions observed for the RS encoder are similar to hot-spot traffic for configurations for 8-port and 16-port switches, i.e., most packets are issued to the first four ports of the switch. By contrast, the packet distributions of the SPARSE application do not present a concentrated load compared to the RS encoder.

As explained in Section V-B1, the switch performance is degraded under traffic patterns like hot-spot because there is only a path available to deliver a packet to a determined destination. Therefore, the performance degradation of the 8-port and 16-port switches using the RS encoder is consistent with the findings of the switch performance using hot-spot traffic.

3) FPGA SYNTHESIS RESULTS

Table 6 summarizes the frequency, area, and power consumption results for a 4, 8, and 16-port switch using the Quartus Prime Lite 21.1 tool for the FPGA Altera Cyclone IV EP4CE115F29C7. The reported values in percentage indicate the amount of FPGA hardware resource utilization. The switch is configured with a 4-flit buffer size and round-robin arbitration. The switch area utilization grows because each port added to the switch consists of an input and output port, plus the extra overhead of the arbiters and the crossbar to manage the new port. Also, to provide minimum logic cells (LCs) utilization, input buffers are mapped on the FPGA embedded memory.

TABLE 6. Hardware cost of the switch for different number of ports.

Ports	Logic Cells	Memory Bits	Max. Freq. (MHz)	Static Power (mW)	Dynamic Power (mW)
4	1312 *(1.1%)*	512 *(0.01%)*	101	98.57	7.63
8	3902 *(3.4%)*	1024 *(0.03%)*	73	98.81	16.27
16	12386 *(10.82%)*	2048 *(0.05%)*	51	99.75	48.66

Notice that the maximum operation frequency decreases as the number of ports increases. This behavior results because the priority variable arbiters of the switch are implemented as a carry chain, which increases the logic levels of the critical path (reported by the timing analyzer tool) when the number of ports is increased.

For accurate dynamic power estimation, post-synthesis netlist files are generated and used to perform gate-level simulations. Realistic toggle rates of the design are extracted from these simulations into a value-change-dump (VCD) file. The testbench used in the gate-level simulations consists of packet sources and sinks. The packet sources inject uniform traffic, and the injection rate is set according to the corresponding saturation throughput obtained in Section V-B1 for 4, 8, and 16-port switch configurations of case C1 (ClkT=ClkSW); this configuration provides an upper bound estimation of power consumption under a worst-case with synthetic traffic. The gate-level simulations are performed in Questa Intel Starter FPGA Edition for the same warm-up and measurement cycles defined in Section V-B1 using a clock of 50 MHz. Only the toggle rates of the measurement period are recorded in the VCD file to get dynamic power consumption when the switch is in a steady state. Then, this information is used in the Power Analyzer Tool included in the Quartus Prime software to get the design's static and dynamic power consumption. The typical operating conditions of the FPGA device selected are used for power estimation.

According to the power results in Table 6, the static power does not increase significantly despite the number of ports configured. This result is expected since the static power consumption is more influenced by the process technology of the FPGA device selected and the operating conditions,

TABLE 7. Hardware cost comparison between ProNoC router, ASIC-based router and 5-port switch.

Component	Logic Cells	Max. Frequency
ProNoC router	1707 (1.49%)	130 MHz
ASIC-based router	1814 (1.58%)	138 MHz
The proposed design	2251 (1.96%)	94 MHz

which determine the leakage in the transistors. On the other hand, dynamic power consumption is influenced by the switching activity of the FPGA resources when processing data. In the case of the switch, while more ports are used, more resources are used to store and forward flits, and therefore, more signal transitions are caused, increasing the dynamic power consumption.

In addition to the above hardware cost results, the proposed design is compared against two Verilog open-source routers presented in the literature to comprehend how close the design is in terms of area regarding other works. The first is a parameterized state-of-the-art router for ASIC-based NoC [88], [89]. The second is the router from ProNoC [90], an open-source tool for prototyping and validation for FPGA-based NoC. Both were selected because they can be configurated with wormhole flow control without VC, which allows making a more fair comparison with the designed wormhole switch configured with five ports.

The NoC unified parameters are 4-flit buffer size, 32-bit payload width, and round-robin arbitration. For routers, dimension order routing is selected due to its simple and inexpensive implementation [6]. Furthermore, because ASIC-based NoC router buffers are not optimized to use the FPGA's embedded memory, a synthesis directive (*ramstyle = "logic"*) was enabled for the ProNoC router and the proposed design. The directive forces the buffers to be synthesized using only LCs, making a fair comparison in area utilization. Table 7 shows the synthesis results of the compared designs.

The synthesis results indicate that the compared designs do not exceed more than 2% of the total LCs available. However, on average, the proposed design uses 27% more LCs than ProNoC and the ASIC-based NoC routers. The maximum operating frequency of the routers outperforms the switch by 42% on average.

The area differences can be explained because the competitors' routers use credit-based link-level flow control, where simple counters keep track of buffer availability. In contrast, the proposed design requires additional logic on each port to perform the required handshake. On the other hand, the frequency differences suggest the possibility of improving frequency by optimizing the critical path, which is affected by the priority variable arbiters. Furthermore, it should be emphasized that a frequency optimization stage was not imposed during the HDL translation. For this reason, an easily parameterized arbiter was chosen, and therefore other implementations were not evaluated.

C. DISCUSSION

1) SWITCH PERFORMANCE CONSIDERATIONS

According to the above switch performance results, using the relationships studied in this work between ClkT and ClkSW, it is recommended that ClkT be equal to or greater than ClkSW for the best throughput. If for some reason, the frequency ClkT must be less than ClkSW, it should be considered that the throughput will be 30% less compared to using a ClkT value equal to or greater than ClkSW.

Different saturation throughputs can be obtained according to the ClkT frequency used. For the clock frequencies shown in Table 3, injecting traffic patterns similar to the permutation is recommended to get a throughput of up to 300 Mbps, which is the best scenario for the switch. Traffic patterns with uniform distribution present throughput of up to 210 Mbps with a 4-port switch, and with 8-port and 16-port switch configurations, throughput decreases up to 10% compared to using a 4-port switch. When traffic patterns similar to hot-spot traffic are injected into the network, it should be considered that a 4-port switch will have the best throughput (up to 200 Mbps), and configurations of more than four ports should be avoided as throughput can decrease up to 50 Mbps.

Considering the overhead due to implementing handshaking, the area cost of the 5-port switch is comparable to the obtained with some state-of-the-art routers. On the other hand, the switch area utilization grows linearly when the number of ports configured increases. Therefore, the maximum port configuration will depend on the features of each technology. For example, with the Cyclone IV device used in this work, if the internal clock of 50 MHz is used, the maximum number of ports supported is 16, with an operating frequency of 51 MHz and 10% of the total FPGA's area occupation. To improve the frequency results, it is suggested to evaluate alternatives to the arbiter based on the carry chain since this type of design increases the critical path when the number of ports increases.

Nevertheless, it should be noted that with a 16-port switch, the frequency drop is approximately 50% compared with the 4-port configuration. Consequently, it is recommended to consider another topology to connect 16 terminals, for example, a 4×4 mesh. In general, if more than 16 ports are needed, alternatives with mesh, torus, and other topologies should be used because they support a higher number of terminals, and their routers have a fixed maximum operating frequency.

As the FPGA is an ideal prototyping tool for designs described at RTL, some situations that would differ from an ASIC should be stated. In an ASIC design, specific constraints, such as input delay, output delay, and clock requirement, are used to optimize a design. Instead, this study only used the default clock timing constraint of 1 GHz. Therefore, the tool made its best effort to get the maximum frequency achievable for each switch configuration. Furthermore, the synthesis tool was used with the default option for the best tradeoff between area and speed.

Regarding the experiment with traces of real workloads, most application execution times are reduced by increasing the number of PBs available to process the application tasks. However, this improvement is at the cost of greater area utilization. Thus, depending on the area budgeted for the NoC, a designer should consider the tradeoff between the hardware cost of the switch and the execution time of an application.

2) SIMULATIONS AND DESIGN VERIFICATION

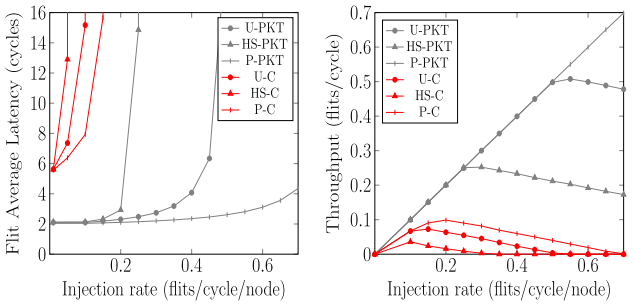
The cycle-level model was used for performance simulations over the RTL model because the simulator OMNet++ provides flexibility in building simulation setups, running parallel simulations, and statistics recollection. Also, other high-level models that interact with the designed cycle-level model can be integrated for simulation. Therefore, it is possible to make simulations where the terminals that inject traffic are described at high levels of abstraction. The communication with models described at lower levels of abstraction is made through interfaces that handle the communication timing.

On the other hand, an advantage of the proposed approach is that through the simulations of each model, an exhaustive verification of the design logic is made. The functionality of the FSMs can be verified before implementing the whole design.

3) PERFORMANCE AT EACH LEVEL OF ABSTRACTION

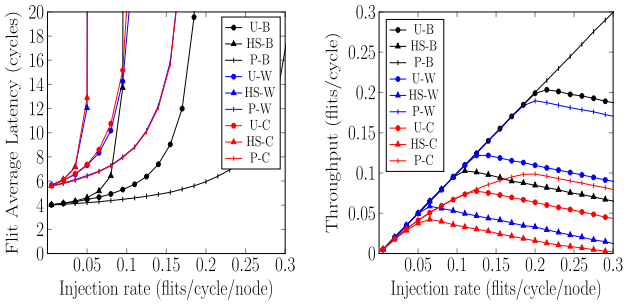
The proposed methodology aims to have a structural approach to derive a microarchitecture according to the NoC functional requirements. Each abstraction level helps the designer progressively integrate different functionalities of the component and then make a block organization to turn it into a microarchitecture model. Therefore, at each abstraction level, it is possible to get a performance approximation related to the cycle-accurate results. Consequently, each new performance approximation will be closer to the one obtained at the cycle level. Latency and throughput curves obtained throughout the switch design in Section IV are presented and discussed. The simulations are performed for an 8-port switch under the same synthetic traffic patterns and NoC configurations explained in Section V-B1. The clock frequencies of case C1 are employed. Performance results for cases C2 and C3 are only relevant at the cycle level where the link signal protocol has already been added.

Fig. 21 shows the latency and throughput results comparing the packet and cycle level models. Due to the simple considerations for the packet level model, its zero-load latency overcomes the cycle level results approximately three times. Consequently, the packet level model accepts more packets than the cycle level model under each traffic pattern, as shown in Fig. 21b. This behavior is expected because the packet processing delay is ideal without using a specialized link protocol. After all, the primary purpose of this model is to provide a first functional understanding of how a NoC component will process a packet format and how the output port is calculated without flit-level details. Therefore, the findings at the packet level should be interpreted with caution. The designer would use these first performance results to understand how a determined topology with a routing algorithm performs over a simple model without considering a flit level congestion. For example, with these results, it can be guessed how each traffic pattern will perform over the following generated models through the design process.



(a) Average Latency (b) Throughput

FIGURE 21. Latency and throughput curves of an 8-port switch at the packet (PKT) and cycle (C) abstraction level. Uniform (U), hot-spot (HS), and permutation (P) synthetic traffic patterns are considered.



(a) Average Latency (b) Throughput

FIGURE 22. Latency and throughput curves of an 8-port switch using flit-level black box (B), white box with the communication protocol (W), and cycle (C) models. Uniform (U), hot-spot (HS), and permutation (P) synthetic traffic patterns are considered.

The subsequent model transformations from the black-box flit-level model present a closer approximation to the cycle level, as is shown in Fig. 22. Both black-box and white-box models at the flit level present the same performance results because they have the same flit delay configuration. Therefore, with this first performance at the flit level, it can be inferred how the component performs assuming a simple backpressure method with similar behavior to the required. When the link signal protocol is added, the impact of the handshake is reflected in the zero load latency with an average increment of two cycles. Thus, there is throughput degradation. Finally, in the cycle-level model, the impact of synchronization of the switch microarchitecture and the handshake with their respective clock sources is reflected, showing the overall impact of the network congestion due to the internal timing delay of the switch.

In particular, the latency and throughput results at different abstraction levels provide an incremental approximation of the designed component performance. However, other metrics could be suitable according to each abstraction level. For example, latency and throughput only show an ideal performance at the packet abstraction level because it only considers the required topology and routing algorithm. Nevertheless, the routing pressure metric [91] could be used at this level for evaluating a routing algorithm because it only requires measuring the number of packets processed by each network channel.

The use of performance metrics through the proposed design process will depend on whether a particular functionality is not defined or cannot be deduced via the initial requirements. Consequently, these analyses can be used as an additional tool to evaluate functional and architectural alternatives of the component when necessary.

VI. CONCLUSION

In this work, it is proposed a NoC components design methodology based on a top-down approach with NoC-oriented abstraction levels, using a switch as a case study. This methodology allowed exploring the switch design space at each level of abstraction according to the specific functional requirements of an SDR system. In this way, different NoC design aspects of the switch were addressed systematically to define its microarchitecture. Implementation results show that the microarchitecture obtained using this approach has an area utilization and frequency operation similar to works in the state-of-the-art. The validation of the designed switch was obtained by evaluating it under different types of synthetic traffic and traces of real workloads. The reported results show that the best performance of the switch is obtained when the applied traffic is similar to permutation traffic with a saturation throughput of 320 Mbps. If the applied traffic is similar to a uniform distribution, the saturation throughput decreases to 200 Mbps. Traffic patterns such as hot-spot should be avoided because the saturation throughput degrades 60% to 80% compared to permutation traffic when the switch is configured with more than four ports. While this work focuses on functional design, the presented methodology makes it possible to identify architecture blocks that could improve a specific metric. Therefore, the designer can evaluate modifications or alternatives on only specific blocks for improving overall performance, which represents a clear advantage over previous approaches.

REFERENCES

[1] A. B. Achballah, S. B. Othman, and S. B. Saoud, "Problems and challenges of emerging technology networks-on-chip: A review," *Microprocess. Microsyst.*, vol. 53, pp. 1–20, Aug. 2017.

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

[3] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Autom. Conf.*, 2001, pp. 684–689.

[4] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI. New Paradigms VLSI Syst. Design. (ISVLSI)*, Apr. 2002, pp. 117–124.

[5] W.-C. Tsai, Y.-C. Lan, Y.-H. Hu, and S.-J. Chen, "Networks on chips: Structure and design methodologies," *J. Electr. Comput. Eng.*, vol. 2012, pp. 1–15, Oct. 2012.

[6] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2004.

[7] J. Flich and D. Bertozzi, *Designing Network On-Chip Architectures in the Nanoscale Era*. London, U.K.: Chapman and Hall/CRC, 2011.

[8] D. Bertozzi, G. Dimitrakopoulos, J. Flich, and S. Sonntag, "The fast evolving landscape of on-chip communication," *Design Autom. Embedded Syst.*, vol. 19, nos. 1–2, pp. 59–76, Mar. 2015.

[9] L. Benini and G. De Micheli, *Networks on Chips: Technology and Tools*. San Mateo, CA, USA: Morgan Kaufmann Publishers, 2006.

[10] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proc. 7th Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2001, pp. 255–266.

[11] P. Poluri and A. Louri, "Shield: A reliable network-on-chip router architecture for chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 3058–3070, Oct. 2016.

[12] K. Parane, B. M. P. Prasad, and B. Talawar, "Design of an adaptive and reliable network on chip router architecture using FPGA," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2019, pp. 1–4.

[13] M. Vinodhini, N. S. Murty, and T. K. Ramesh, "Transient error correction coding scheme for reliable low power data link layer in NoC," *IEEE Access*, vol. 8, pp. 174614–174628, 2020.

[14] N. K. Baloch, M. I. Baig, and M. Daneshtalab, "Defender: A low overhead and efficient fault-tolerant mechanism for reliable on-chip router," *IEEE Access*, vol. 7, pp. 142843–142854, 2019.

[15] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, P. Kundu, and N. Borkar, "A 2 Tb/s 6 × 4 mesh network with DVFS and 2.3 Tb/s/W router in 45 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2010, pp. 79–80.

[16] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[17] Z. Wang, S. Ma, L. Huang, M. Lai, and W. Shi, *Networks-On-Chip*. Oxford, U.K.: Morgan Kaufmann, 2015.

[18] L. Huang, Z. Wang, and N. Xiao, "VBON: Toward efficient on-chip networks via hierarchical virtual bus," *Microprocessors Microsyst.*, vol. 37, no. 8, pp. 915–928, Nov. 2013.

[19] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 420–431, May 2005.

[20] Y. S. Yang, H. Deshpande, G. Choi, and P. V. Gratz, "SDPR: Improving latency and bandwidth in on-chip interconnect through simultaneous dual-path routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 3, pp. 545–558, Mar. 2018.

[21] A. Bose and P. Ghosal, "Switching at flit level: A congestion efficient flow control strategy for network-on-chip," in *Proc. 28th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. (PDP)*, Mar. 2020, pp. 319–322.

[22] A. Psarras, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "ShortPath: A network-on-chip router with fine-grained pipeline bypassing," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3136–3147, Oct. 2016.

[23] Y. Li and A. Louri, "ALPHA: A learning-enabled high-performance network-on-chip router design for heterogeneous manycore architectures," *IEEE Trans. Sustain. Comput.*, vol. 6, no. 2, pp. 274–288, Apr. 2021.

[24] W.-T. Wu and A. Louri, "A methodology for cognitive NoC design," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 1–4, Jan. 2016.

[25] J. Yin, S. Sethumurugan, Y. Eckert, C. Patel, A. Smith, E. Morton, M. Oskin, N. E. Jerger, and G. H. Loh, "Experiences with ML-driven design: A NoC case study," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 637–648.

[26] K. Wang and A. Louri, "CURE: A high-performance, low-power, and reliable network-on-chip design using reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2125–2138, Sep. 2020.

[27] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proc. 31st Annu. Int. Symp. Comput. Archit.*, 2004, pp. 188–197.

[28] R. Mullins, A. West, and S. Moore, "The design and implementation of a low-latency on-chip network," in *Proc. Asia South Pacific Conf. Design Autom.*, 2006, pp. 164–169.

[29] A. Kumary, P. Kunduz, A. P. Singhx, L.-S. Pehy, and N. K. Jhay, "A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65 nm CMOS," in *Proc. 25th Int. Conf. Comput. Design*, Oct. 2007, pp. 63–70.

[30] H. Matsutani, M. Koibuchi, H. Amano, and T. Yoshinaga, "Prediction router: Yet another low latency on-chip router architecture," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Feb. 2009, pp. 367–378.

[31] A. Psarras, J. Lee, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "PhaseNoC: Versatile network traffic isolation through TDM-scheduled virtual channels," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 844–857, May 2016.

[32] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. Iyer, and C. R. Das, "Design of a dynamic priority-based fast path architecture for on-chip interconnects," in *Proc. 15th Annu. IEEE Symp. High-Perform. Interconnects (HOTI)*, Aug. 2007, pp. 15–20.

[33] R. Tamhankar, S. Murali, S. Stergiou, A. Pullini, F. Angiolini, L. Benini, and G. De Micheli, "Timing-error-tolerant network-on-chip design methodology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1297–1310, Jul. 2007.

[34] S. M. Hassan and S. Yalamanchili, "Centralized buffer router: A low latency, low power router for high radix NOCs," in *Proc. 7th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, Apr. 2013, pp. 1–8.

[35] I. Seitanidis, A. Psarras, G. Dimitrakopoulos, and C. Nicopoulos, "ElastiStore: An elastic buffer architecture for network-on-chip routers," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[36] G. Michelogiannakis and W. J. Dally, "Elastic buffer flow control for on-chip networks," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 295–309, Feb. 2013.

[37] A. K. Kodi, A. Sarathy, and A. Louri, "IDEAL: Inter-router dual-function energy and area-efficient links for Network-on-Chip (NoC) architectures," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2008, pp. 241–250.

[38] A. Psarras, S. Moisidis, C. Nicopoulos, and G. Dimitrakopoulos, "Networks-on-chip with double-data-rate links," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 12, pp. 3103–3114, Dec. 2017.

[39] D. U. Becker, N. Jiang, G. Michelogiannakis, and W. J. Dally, "Adaptive backpressure: Efficient buffer management for on-chip networks," in *Proc. Int. Conf. Comput. Des. (ICCD)*, Sep. 2012, pp. 419–426.

[40] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "ViChaR: A dynamic virtual channel regulator for network-on-chip routers," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 333–346.

[41] A. C. Pinheiro, J. A. N. Silveira, D. A. B. Tavares, F. G. A. Silva, and C. A. M. Marcon, "Optimized fault-tolerant buffer design for network-on-chip applications," in *Proc. IEEE 10th Latin Amer. Symp. Circuits Systems (LASCAS)*, Feb. 2019, pp. 217–220.

[42] M. Katta, T. K. Ramesh, and J. Plosila, "SB-Router: A swapped buffer activated low latency network-on-chip router," *IEEE Access*, vol. 9, pp. 126564–126578, 2021.

[43] J. Fang, Z. Chang, and D. Li, "Exploration on routing configuration of HNoC with intelligent on-chip resource management," *IEEE Access*, vol. 8, pp. 12117–12129, 2020.

[44] A. Abbas, M. Ali, A. Fayyaz, A. Ghosh, A. Kalra, S. U. Khan, M. U. S. Khan, T. De Menezes, S. Pattanayak, A. Sanyal, and S. Usman, "A survey on energy-efficient methodologies and architectures of network-on-chip," *Comput. Electr. Eng.*, vol. 40, no. 8, pp. 333–347, Nov. 2014.

[45] C. Silvano, M. Lajolo, and G. Palermo, *Low Power Networks-on-Chip*. Boston, MA, USA: Springer, 2011.

[46] K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-performance SoC design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 2, pp. 148–160, Feb. 2006.

[47] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-36)*, Dec. 2003, pp. 105–116.

[48] A. Agarwal, G. L. Hamza-Lup, and T. M. Khoshgoftaar, "A system-level modeling methodology for performance-driven component selection in multicore architectures," *IEEE Syst. J.*, vol. 6, no. 2, pp. 317–328, Jun. 2012.

[49] K. Bergman, L. P. Carloni, A. Biberman, J. Chan, and G. Hendry, *Photonic Network-on-Chip Design* (Integrated Circuits and Systems), vol. 68. New York, NY, USA: Springer, 2014.

[50] A. Ben and S. Ben, "A survey of network-on-chip tools," *Int. J. Adv. Comput. Sci. Appl.*, vol. 4, no. 9, pp. 61–67, 2013.

[51] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.

[52] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. De Micheli, and L. Benini, "NoC design and implementation in 65 nm technology," in *Proc. 1st Int. Symp. Networks Chip (NOCS)*, May 2007, pp. 273–282.

[53] Y. Chen, S. T. Gurumani, Y. Liang, G. Li, D. Guo, K. Rupnow, and D. Chen, "FCUDA-NoC: A scalable and efficient network-on-chip implementation for the CUDA-to-FPGA flow," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 6, pp. 2220–2233, Jun. 2016.

[54] K. Helal, S. Attia, H. A. H. Fahmy, T. Ismail, Y. Ismail, and H. Mostafa, "Dual split-merge: A high throughput router architecture for FPGAs," *Microelectron. J.*, vol. 81, pp. 51–57, Nov. 2018.

[55] J. Kim, "Low-cost router microarchitecture for on-chip networks," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture (Micro-42)*, 2009, pp. 255–266.

[56] J. Kim and H. Kim, "Router microarchitecture and scalability of ring topology in on-chip networks," in *Proc. 2nd Int. Workshop Netw. Chip Architectures*, 2009, pp. 5–10.

[57] H. Kim, G. Kim, H. Yeo, J. Kim, and S. Maeng, "Design and analysis of hybrid flow control for hierarchical ring network-on-chip," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 480–494, Feb. 2016.

[58] A. Roca, J. Flich, F. Silla, and J. Duato, "A low-latency modular switch for CMP systems," *Microprocessors Microsyst.*, vol. 35, no. 8, pp. 742–754, Nov. 2011.

[59] A. Roca, C. Hernández, J. Flich, F. Silla, and J. Duato, "Silicon-aware distributed switch architecture for on-chip networks," *J. Syst. Archit.*, vol. 59, no. 7, pp. 505–515, Aug. 2013.

[60] A. T. Tran and B. M. Baas, "RoShaQ: High-performance on-chip router with shared queues," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, Oct. 2011, pp. 232–238.

[61] Y. Ben-Itzhak, I. Cidon, A. Kolodny, M. Shabun, and N. Shmuel, "Heterogeneous NoC router architecture," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2479–2492, Sep. 2015.

[62] R. S. Ramanujam, V. Soteriou, B. Lin, and L.-S. Peh, "Design of a high-throughput distributed shared-buffer NoC router," in *Proc. 4th ACM/IEEE Int. Symp. Networks Chip*, May 2010, pp. 69–78.

[63] C. Das, M. Yousif, V. Narayanan, D. Park, C. Nicopoulos, and J. Kim, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proc. 33rd Int. Symp. Comput. Archit. (ISCA)*, 2006, pp. 4–15.

[64] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proc. 42nd Design Autom. Conf.*, 2005, pp. 559–564.

[65] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das, "A novel dimensionally-decomposed router for on-chip communication in 3D architectures," in *Proc. 34th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2007, pp. 138–149.

[66] P. Abad, V. Puente, J. A. Gregorio, and P. Prieto, "Rotary router: An efficient architecture for CMP interconnection networks," in *Proc. 34th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2007, pp. 116–125.

[67] G. Michelogiannakis, D. Pnevmatikatos, and M. Katevenis, "Approaching ideal NoC latency with pre-configured routes," in *Proc. 1st Int. Symp. Networks Chip (NOCS)*, May 2007, pp. 153–162.

[68] G. Chen, M. A. Anders, H. Kaul, S. K. Satpathy, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, V. De, and S. Borkar, "A 340 mV-to-0.9 V 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16 × 16 network-on-chip in 22 nm tri-gate CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 59–67, Jan. 2015.

[69] W. Singh and S. Deb, "Energy- and performance-aware router design for chip multiprocessors," in *Proc. IEEE Int. Symp. Smart Electron. Syst. (iSES) (Formerly iNiS)*, Dec. 2019, pp. 316–319.

[70] L. Wang, X. Wang, and Y. Wang, "An approximate bufferless network-on-chip," *IEEE Access*, vol. 7, pp. 141516–141532, 2019.

[71] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A low-complexity bufferless deflection router," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 144–155.

[72] X. Xiang, P. Sigdel, and N.-F. Tzeng, "Bufferless network-on-chips with bridged multiple subnetworks for deflection reduction and energy savings," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 577–590, Apr. 2020.

[73] M. Rivera-Acosta, S. Ortega-Cisneros, and J. Rivera, "Automatic tool for fast generation of custom convolutional neural networks accelerators for FPGA," *Electronics*, vol. 8, no. 6, p. 641, Jun. 2019.
[74] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits Syst. Mag.*, vol. 4, no. 2, pp. 18–31, Sep. 2004.
[75] J. L. Vazquez-Avila, R. Sandoval-Arechiga, B. I. Gea-Garcia, R. Parra-Michel, and Mario-Siller, "Unconventional signal processing architecture for reconfigurable on-chip communication systems," in *Proc. IEEE 6th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2015, pp. 1–4.
[76] W. Hussain, R. Airoldi, H. Hoffmann, T. Ahonen, and J. Nurmi, "Design of an accelerator-rich architecture by integrating multiple heterogeneous coarse grain reconfigurable arrays over a network-on-chip," in *Proc. IEEE 25th Int. Conf. Appl.-Specific Syst., Archit. Processor*, Jun. 2014, pp. 131–138.
[77] S. Nouri, W. Hussain, and J. Nurmi, "Evaluation of a heterogeneous multicore architecture by design and test of an OFDM receiver," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3171–3187, Nov. 2017.
[78] C. Zhang, L. Liu, D. Marković, and V. Öwall, "A heterogeneous reconfigurable cell array for MIMO signal processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 3, pp. 733–742, Mar. 2015.
[79] X. Fang and S. Chen, "The design and algorithm mapping of a heterogeneous multi-core processor for SDR," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Nov. 2008, pp. 1086–1089.
[80] A. Chun, "Key lessons from the scalable communications core: A reconfigurable wireless baseband," *IEEE Commun. Mag.*, vol. 48, no. 12, pp. 101–109, Dec. 2010.
[81] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *Proc. IEEE*, vol. 85, no. 3, pp. 366–390, Mar. 1997.
[82] A. Jantsch and I. Sander, "Models of computation and languages for embedded system design," *IEE Proc. Comput. Digit. Techn.*, vol. 152, no. 2, pp. 114–129, Mar. 2005.
[83] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proc. 1st IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2003, pp. 19–24.
[84] A. Varga. (2001). *OMNeT++ Discrete Event Simulation System*. [Online]. Available: https://omnetpp.org/
[85] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Des. Test Comput.*, vol. 28, no. 5, pp. 23–35, Sep./Oct. 2011.
[86] C. Iordanou, V. Soteriou, K. Aisopos, and E. Kakoulli, "Hermes: Architecting a top-performing fault-tolerant routing algorithm for networks-on-chips," in *Proc. 8th IEEE/ACM Int. Symp. Networks Chip (NoCS)*, Sep. 2014, pp. 178–179.
[87] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC traffic suite based on real applications," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2011, pp. 66–71.
[88] D. U. Becker, "Efficient microarchitecture for network-on-chip routers," Ph.D. dissertation, Stanford, CA, USA, 2012.
[89] D. U. Becker. (2022). *Stanford Digital Repository*. [Online]. Available: https://library.stanford.edu/research/stanford-digital-repository
[90] A. Monemi, J. W. Tang, M. Palesi, and M. N. Marsono, "ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform," *Microprocessors Microsyst.*, vol. 54, pp. 60–74, Oct. 2017.
[91] M. Tang, X. Lin, and M. Palesi, "Routing pressure: A channel-related and traffic-aware metric of routing algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 891–901, Mar. 2015.

V. A. DELGADO-GALLARDO (Member, IEEE) was born in Mexico City, Mexico, in 1991. He received the B.Sc. degree in electronics and communications from Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME), Instituto Politecnico Nacional (IPN), Mexico City, in 2013, and the M.Sc. degree in electrical engineering specializing in communications from Centro de Investigación y Estudios Avanzados (CINVESTAV), IPN, Guadalajara, in 2018, where he is currently pursuing the Ph.D. degree. His research interests include the design and implementation of on-chip networks.

R. SANDOVAL-ARECHIGA (Member, IEEE) was born in Aguascalientes, Mexico, in 1979. He received the B.Sc. degree in communications and electronics from the Autonomous University of Zacatecas, in 2002, the M.Sc. degree in electrical engineering specializing in communications from CINVESTAV-IPN, Mexico City, in 2006, and the Ph.D. degree in electrical engineer specialized in communications from CINVESTAV-IPN, Guadalajara, in 2016. He has collaborated with several Mexican companies and government agencies. He is currently the Director of the Center of Research, Innovation and Development in Telecommunications, Autonomous University of Zacatecas. His research interests include networking, on-chip design and analysis, and modeling of complex systems.

R. PARRA-MICHEL (Member, IEEE) was born in Guadalajara, Mexico, in 1973. He received the B.Sc. degree in electronics and communications from the University of Guadalajara, in 1996, the M.Sc. degree in electrical engineering specializing in communications from CINVESTAV-IPN, Guadalajara, in 1998, and the Ph.D. degree in electrical engineering specialized in digital signal processing for communications from CINVESTAV-IPN, Mexico City, in 2003. He has collaborated with several companies and institutions either in academic or technology projects, such as Siemens, Lucent, Mabe, Mixbaal, Hewlett-Packard, and Intel. He is currently the Director of the Guadalajara Unit, CINVESTAV-IPN, and the Head of the Wireless Communication Group. His research interests include modeling, simulation, estimation and equalization of communication channels, and digital implementation of DSP algorithms for communication systems and on-chip networks.