Design and Multi-Abstraction Level Evaluation of a NoC Router for Mixed-Criticality Real-Time Systems

Mourad Dridi, Université de Bretagne Occidentale, Lab-STICC UMR CNRS 6285, Brest, France
Stéphane Rubini, Université de Bretagne Occidentale, Lab-STICC UMR CNRS 6285, Brest, France
Mounir Lallali, Université de Bretagne Occidentale, Lab-STICC UMR CNRS 6285, Brest, France
Martha Johanna Sepúlveda Flórez, Technical University of Munich, Germany
Frank Singhoff, Université de Bretagne Occidentale, Lab-STICC UMR CNRS 6285, Brest, France
Jean-Philippe Diguet, CNRS, Lab-STICC UMR CNRS 6285, Lorient, France

A Mixed Criticality System (MCS) combines real-time software tasks with different criticality levels. In a MCS, the criticality level specifies the level of assurance against system failure. For high-critical flows of messages, it is imperative to meet deadlines, otherwise the whole system might fail, leading to catastrophic results, like, loss of life or serious damage to the environment. In contrast, low-critical flows may tolerate some delays.

Furthermore, in MCS, flow performances such as the Worst Case Communication Time (WCCT) may vary depending on the criticality level of the applications. Then, execution platforms must provide different operating modes for applications with different levels of criticality. To conclude, in Network-On-Chip (NoC), sharing resources between communication flows can lead to unpredictable latencies and subsequently turns the implementation of MCS in many-core architectures challenging.

In this article, we propose and evaluate a new NoC router to support MCS based on an accurate WCCT analysis for high-critical flows. The proposed router, called **DAS** (**D**ouble **A**rbiter and **S**witching router), jointly uses *Wormhole* and *Store And Forward* communication techniques for low and high-critical flows respectively. It ensures that high-critical flows meet their deadlines while maximizing the bandwidth remaining for the low-critical flows. We also propose a new method for high-critical communication time analysis, applied to *Store And Forward* switching mode with virtual channels. For low-critical flows communication time analysis, we adapt an existing wormhole communication time analysis with share policy to our context.

The second contribution of this paper is a multi-abstraction level evaluation of DAS. We evaluate the communication time of flows, the system mode change, the cost and 4 properties of DAS. Simulations with a cycle-accurate SystemC NoC simulator show that, with a 15% network use rate, the communication delay of high-critical flows is reduced by 80% while communication delay of low-critical flow is increased by 18% compared to solutions based on routers with multiple virtual channels. For 10% of network interferences, using system mode change, DAS reduces the high-critical communication delays about 66%. We synthesize our router with a 28nm SOI technology and show that the size overhead is limited of 2.5% compared to the solution based on virtual channel router. Finally, we applied model checking verification techniques to automatically prove several DAS properties required by critical systems designers.

Additional Key Words and Phrases: Network-On-Chip, Mixed-Criticality System, Wormhole, Store and Forward, NoC Router, Communication Time, Virtual Channel Router, IF-Language, SHoC

1. INTRODUCTION

Many-core architectures allow multiple applications to run simultaneously on the same chip. These applications communicate with each other by exchanging messages through the communication infrastructure. NoCs are used to support the high communication demands of many core architectures. They integrate routers and links to allow the on-chip communications. NoCs provide communication parallelism with high bandwidth and modularity. The expected processing power and high communication bandwidth of NoCs architectures allow designers to deploy many computation intensive functions. For example, in a drone application, designers will need to deploy a classical flight control software but also computation intensive functions such as image and video processing.

In the context of such systems, the criticality of these applications can be different. Burns et al. [Burns and Davis 2017] define the criticality as *a designation of the level of* *assurance against failure needed for a system component*. This kind of systems, characterized by two or more distinct levels of criticality, is called Mixed Criticality Systems (MCS) [Burns and Davis 2017].

In MCS, hard real-time and soft real-time applications share the same hardware platform and exchange messages through a common communication infrastructure. Hard real-time applications have very stringent communication service requirements. Longitudinal flight controller [Pagetti et al. 2014] is an example of hard real-time application. It is imperative to meet deadlines otherwise the whole system might fail, leading to catastrophic results, like, loss of life or serious damage to the environment. In contrast, soft real-time applications can tolerate some missed deadlines. Video encoder is an example of soft real-time applications. So, in this article, we will consider two criticality levels of communication flows: high-critical flows and low-critical flows.

MCSs operate in different modes in order to ensure that high-critical flows always meet their deadlines whilst maximizing the use of shared resources. A system starts in the normal mode, which is the operating mode when all communication timing requirements are met [Burns and Davis 2017]. In this article, we focus on Worst Case Communication time (WCCT) which is the maximum latency a message for a given flow will take from the sender to the receiver task. A mode change occurs from the normal mode to the degraded mode if at least one message is delayed for a longer time than is acceptable in the current mode, i.e., when the message WCCT is longer than its deadline. When switching to the degraded mode, messages may be suspended or aborted depending on their criticality level. In degraded modes, some or all low-critical flows are suspended in order to give more resources to high-critical flows and to reduce high-critical flows WCCT. So a MCS can be either in a normal mode or in a degraded mode depending on unpredictable latency due to NoC shared resources.

To sum up, we focus our work on the deployment of MCS on NoC-based architectures. Applications with different levels of criticality exchange messages through the communication infrastructure. Sharing resources such as routers, links and buffers can lead to different unpredictable latencies due to direct interferences and indirect interferences [Shi and Burns 2008]. Subsequently, the deployment of MCS on NoCs is a real challenge.

The key component of a NoC is the router, due to its impact on the performance and on the cost of the chip. Several NoC router architectures have been proposed in order to minimize delays of messages and to satisfy the timing requirements of hard and soft real-time applications. However, none of them is able to handle simultaneously high and low-critical flows. Time-Division-Multiplexing (TDM) routers provide a solution for flows contention in NoC using time slot reservation [Liu et al. 2015]. TDM routers are not suitable for MCS because they lead to low throughput for low-critical flows while providing highly deterministic communication time for high-critical flows. Conversely, Virtual Channel (VC) routers reduce the latencies of flows and increase the network throughput [Kavaldjiev et al. 2004]. However, VC routers also are not suitable for MCS: while providing high throughput for low-critical flows, they lead to too pessimistic WCCT of high-critical flows [Indrusiak et al. 2016].

An on-chip communication architecture for MCS must provide different operating modes for applications with different levels of criticality deployed on the same network, in order to reduce the overestimation of WCCT of high-critical flows while improving the throughput for low-critical flows.

We propose and evaluate a new router called DAS (Double Arbiter and Switching). It has been designed to efficiently deploy MCS applications over NoCs, and it ensures the timing constraints for high-critical flows while limiting the bandwidth reservation for them. In this article, we describe the design of DAS, and we present an accurate WCCT analysis suitable for this communication architecture.

DAS is based on N+1 virtual channels: N VCs are dedicated to the communication of the high-critical flows and the last one is shared by low-critical flows. To enforce MCS requirements, DAS implements automatic mode changes, 2 levels of preemption, 2 flow control techniques and 2 stages of arbitration. Operating in different modes and combination of SAF and virtual channels lead to a deterministic WCCT for the high-critical flows.

In this work, we assume a mapping of the application tasks which ensures that less than N high-critical flows share a given physical link in the NoC. We also assume that high-critical flows are characterized by small packet sizes (e.g., control commands), while low-critical flows are characterized by larger ones (e.g., multimedia packets).

The second contribution of this paper is the multi-abstraction level evaluation of DAS features. We have used multiple techniques based on different abstraction models in order to evaluate DAS expected properties. We considered cycle accurate simulations, formal verification and hardware synthesis. We define four properties DAS has to comply with in order to formalize an analytic model of the WCCT. Then, those properties, the WCCT and also the efficiency of system mode change are validated by using cycle-accurate System C simulations, model checking with the IFx toolset and a 28nm SOI technology hardware synthesis. The conjunction of these evaluations strongly underscores the interest of our proposal for supporting the deployment of MCS on NoC-based architectures.

The remainder of the article is organized as follows. First, we discuss related works in section 2. Then, we provide an overview of MCS and VC routers, and we present IF language the IF toolset in section 3. In Section 4, we detail the architecture of the proposed router. The mode change policy and the subsequent WCCT analysis are explained in Section 5. DAS properties validation with model checking, cost evaluation and simulations are presented in Section 6 and compared to those of a VC router. The Section 7 concludes the paper.

2. RELATED WORK

In this section, we propose an analysis of works related to NoC routers supporting MCS and highlight their main differences compared to DAS. Moreover, we present formal verifications work applied to NoCs designs.

2.1. NoCs supporting MCS

In recent years, there has been a growing interest to design NoC routers for Mixed-criticality applications. Several routers architectures and protocols have been proposed to deal with the trade-off between resource sharing and separation of different criticality levels.

Tobuschat et al. have proposed a protocol in order to deploy MCS over NoCs architectures. Their protocol, called IDAMC [Tobuschat et al. 2013], maximizes the bandwidth given to low-critical flows whilst ensuring timing constraints for high-critical flows.

WPMC [Burns et al. 2014] is another protocol applied to Wormhole virtual channels NoCs in order to running MCS over NoCs. This protocol is improved in [Indrusiak et al. 2015].

Previous protocols [Tobuschat et al. 2013], [Burns et al. 2014] and [Indrusiak et al. 2015] use Wormhole policy with flit-level priority preemption. With those protocols, high-critical flows can be preempted by other highly priority flows in flit-level. In addition, each virtual channel can be shared by many high-critical flows. This configuration leads to a too pessimistic WCCT and an over reservation of bandwidth for high-critical flows, which will limit the low-critical communications [Indrusiak et al. 2016].

Ahmadian et al. have proposed Time-Triggered Extension layer (TTEL) for NoC interfaces [Ahmadian and Obermaisser 2015]. With this layer, the NoC is capable of

Table I. Related Work

NoC	Topology	Routing Algorithm	Switching technique
DSPIN	mesh	XY	Wormhole
Æthereal	Indirect network	Contention-free source routing	TDM
TRIPS(OCN)	mesh	YX	Wormhole
TERA flops	mesh	YX and Source routing	Wormhole
NoC with DAS	mesh	XY	Wormhole and SAF

NoC	Different type of traffic	Pessimistic Worst Case Communication time	Dynamic resources allocation	Suitable for MCS
DSPIN	Yes: best effort and guaranteed service	Yes: Wormhole with virtual channels	Yes: Priority-based flit level preemption	No: Pessimistic communication time scheduling
Æthereal	Yes: best effort and guaranteed service	No: TDM	No: TDM-based arbitration	No: No flit-level preemption
TRIPS (OCN)	No	Yes: Wormhole with virtual channels	No: Fair arbitration	No: support one type of traffic only
TERA flops	No	Yes: Wormhole with virtual channels	No: Fair arbitration	No: support one type of traffic only
NoC with DAS	Yes: high and low-critical flows	No: SAF for high-critical flow	Yes: Criticality-based flit level preemption	Yes

supporting multiple types of communication such as Time-triggered messages, Rate-constrained messages and Best-effort messages. At the transport layer, in order to resolve the contention between messages of different traffic types, TTEL imposes the periods and the phases for time-triggered messages when the Best-effort messages use only the remaining available bandwidth. However, at the network layer, TTEL assigns the highest priority to the time-triggered messages and the lower priorities for other traffic types. Source-based routing is required to use TTEL. Using source routing algorithm, the source node makes all decisions about the routing path of the packet which complicates the prediction of the worst case communication time. In addition, low-critical flows cannot share the same path with a high-critical flow which makes TTEL not suitable for MCS.

Tobuschat et al. have proposed a run-time configurable NoC which supports safety-critical and best-effort traffic [Tobuschat and Ernst 2017a]. It gives priority to best-effort traffic over critical traffic, while monitoring is used to change the priority during at run-time. The flit header is extended with an additional field which hold the slack information. According to the slack information, priority change is managed. In this approach, they do not take into account the criticality mode of MCS. Furthermore, for long packets, adding the slack information to each flit may lead to additional delays. Finally, the use of Wormhole policy with shared priority and flit-level preemption make complex the prediction of communication delays.

Several papers show that Wormhole switching mode is suitable for high-critical applications.

[Rahmati et al. 2013] presents methods to calculate worst case bandwidth and latency bounds for Real-Time traffic on wormhole NoCs with arbitrary topology. The proposed methods apply to best-effort NoC architectures with a round-robin arbitration.

In this work, authors do not consider flit-level priority preemption. Again, according to the MCS definition we assume and considering round-robin arbitration and no priority preemption make this analysis not suitable for us.

[Tobuschat and Ernst 2017b] proposes a real-time communication analysis for best-effort NoCs with bounded size buffers and backpressure. Backpressure occurs when the queues in the routers is overflowing. Backpressure constitutes a significant problem in systems with a shared channel. The authors consider a basic NoC with shared virtual channels and without flit-level priority preemption. The proposed analysis can be used for MCS running NoCs with shared virtual channels. We note here that using DAS, high-critical flows do not share the same virtual channel subsequently we do not need to consider backpressure in our analysis.

[Panic et al. 2016] analyzes the contention in wormhole based NoC. It proposes a new metric to measure the worst case communication time: the worst-case contention delay (WCD). WCD takes into account the pipelined behavior of wormhole based NoC. It captures the impact of wormhole switching mode on worst case execution time. [Panic et al. 2016] proposes an analytical model that computes the WCD bounds for several wormhole based NoC design which help to deploy hard real time application on NoC architectures. However, in our analysis, we have chosen to work with the metric WCCT proposed in [Shi 2009]

In this paper, we propose a solution which combines SAF and Wormhole. Using SAF for high-critical flows facilitates the computation of WCTT while using Wormhole with flit level preemption for low-critical flows allow us to increase the use rate of the network by low-critical flows.

There are also many NoC architectures for real-time applications, but none of them are suitable for MCS. In Table I, we compare DAS based NoC with several existing NoC architectures DSPIN [Miro-Panades et al. 2008], Æthereal [Goossens et al. 2005], TRIPS(OCN) and Teraflops [Ma et al. 2014]. For each NoC, we present the main characteristics (topology, routing algorithm and switching technique), the expected behavior and the suitability for MCS.

As shown in Table I, DSPIN and Æthereal support two different types of traffic: best effort and guaranteed service, while TRIPS(OCN) and Teraflops are limited to only one type of traffic.

For DSPIN, TRIPS(OCN) and Teraflops, using Wormhole virtual channel switching mode makes the WCCT pessimistic, while TDM-switching allows more accurate WCCT in Æthereal.

TRIPS(OCN) and Teraflops use fair arbitration. TDM-Based arbitration is used in Æthereal. For those NoCs, there is no dynamic resources allocation based on priority or criticality of flows, while DSPIN uses priority-based flit level preemption to ensure timing constraint of guaranteed service.

TRIPS(OCN) and Teraflops do not accept different types of traffic. Wormhole VC with priority-based flit-level preemption make the computation of WCTT more pessimistic and difficult. Subsequently they make DSPIN not suitable for MCS.

Here, we can conclude, from Table I, that none of those NoCs are suitable for MCS.

2.2. Formal verification

Last part of this section is about formal verifications in the context of NoC architecture design.

[Gholami and Sarjoughian 2017] proposes extensions to Discrete EVent System Specification (DEVS) modeling framework for NOC router component model checking by using constrained DEVS models.

Using Heterogeneous Protocol Automota (HPA), Palaniveloo et al. design a formal model of the existing HERMES NoC router architecture [Moraes et al. 2004] and its communication scheme [Palaniveloo and Sowmya 2011]. Simple Promela Interpreter (SPIN) tool is used for modeling and verifying in order to check the functional properties of the communication architecture.

Four crucial properties of an NoC router, namely, mutual exclusion, starvation freedom, deadlock freedom, and conditions for traffic congestions have been verified in [Chen et al. 2010]. In this work, authors use a formal verification model checking tool called State Graph Manipulators (SGM) to perform such verifications for the bidirectional channel Network-on-Chip (BiNoC).

Using formal methods, [Zaman 2015] proposes a functional and performance analysis of two types of NoC: circuit switched NoC and packet switched NoC. HERMES is the chosen packet-switched NoC, while, the Programmable NoC (PNoC) [Zaman 2015] was chosen as a circuit switched NoC. Properties such as mutual exclusion, starvation freedom, deadlock and livelock have been verified using the SPIN model checker.

Previous works verify existing NoCs with formal methods, but none of them focused on deployment of MCS over NoCs. Thus, NoC properties related to MCS, such as the preemption in flit level between flows with different levels of criticality, were never investigated. The verification of these properties is absolutely necessary in order to run MCS over NoCs. In this paper, we focused on this shortfall. First, we gave a formal specification in IF Language of DAS. Second, a formal validation of a NoC router supporting MCS was performed. We actually investigated DAS properties related to MCS such as flit-level preemption.

3. BACKGROUND

This section presents the necessary background on NoC routers and MCS to understand the contributions proposed in the next sections. IF language and the IF toolset are also introduced in this section.

3.1. Virtual Channel NoC Router

The key component in a NoC is the router. The parameters of a router are very important as they can modify the power consumption and performances [Jetly 2013].

Several NoC router architectures have been proposed such as TDM router, virtual channel (VC) router and dynamic router [Ma et al. 2014]. In the sequel, we focus on VC router, which is the most used to reduce the communication latency of flows. It is a state of the art solution to allocate communications resources to different flows according to priorities, so a candidate to manage MCS flows.

Fig. 1 illustrates the major components of a VC router. A VC presents an unidirectional logical connection between two routers multiplexed with other virtual channels across the physical channel. As shown in Fig. 1, the router is composed of input and output ports, a routing logic, a VC allocator, a switch allocator and a crossbar [Kavaldjiev et al. 2004].

3.1.1. Routing logic. It specifies the path of the packet to travel from the source node to the destination node. For routing algorithms, there are some properties which are required for interconnection network such as connectivity and deadlock avoidance. Many routing algorithms have been proposed such as XY, source routing and DyXY [Ma et al. 2014]. In this work, we consider XY routing algorithm, which is a classical and simple deadlock free solution.

3.1.2. VC allocator, crossbar and arbiters. The VC allocator assigns an available output channel to a new packet located in one of the input VC buffers. In order to allocate incoming packets to output channels, an arbitration between all packets requesting the same output channel is required. After this arbitration and if the requested link

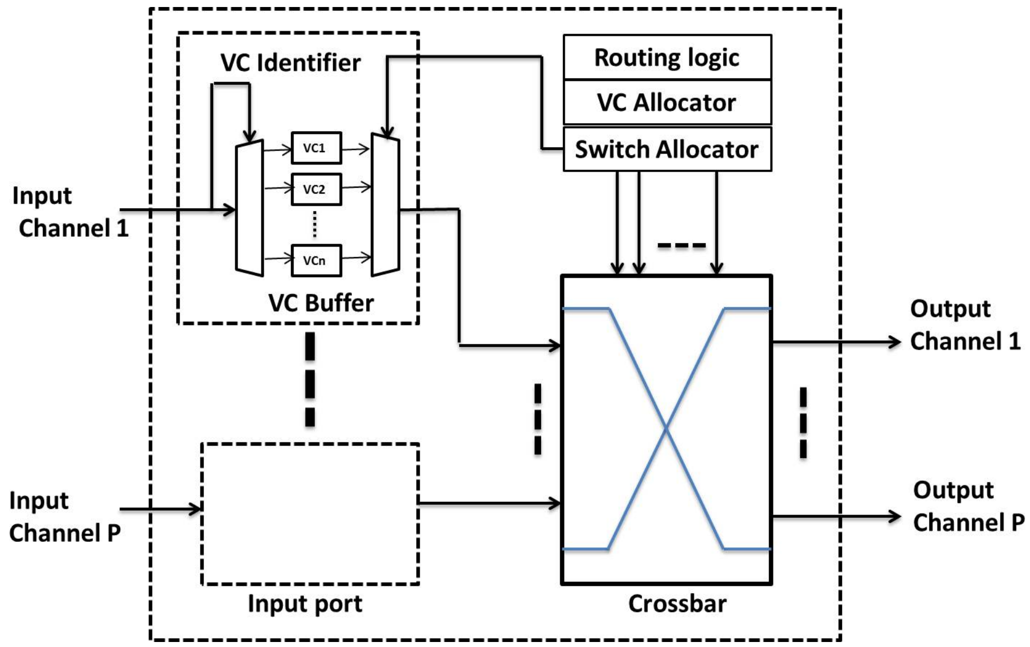


Fig. 1. Virtual Channel Router. The router has P input ports and P output ports, supporting n virtual channels per port.

is available, the input virtual channel will request an access to the selected output channel via the router crossbar.

Several arbitration mechanisms have been used in NoC routers [Jain et al. 2015]. Round-robin and priority-based are 2 examples of these mechanisms: (1) *round-robin arbiter* gives the lowest priority to the last served request in the next arbitration, (2) *priority-based arbiter* chooses one packet from many requests based on their priority.

3.1.3. Switch Allocator. It performs a matching between incoming packets and output ports, and generates the required crossbar control signals. Moreover, the switching mode determines how a packet is allocated to buffers and channels and when it will receive service. For the *Store And Forward* mode (SAF), each switch waits for the full packet to arrive before sending it to the next router [Jetly 2013]. SAF has a huge impact on the buffer size and communication time which depend on packet size. For the *Wormhole* mode, the packet is divided into a number of fixed size flits [Shi 2009]. The packet is split into an header flit, one or several body flits and a tail flit. The header flit stores the routing information and is used to build the route. As the header flit moves ahead along the selected path, the remaining flits follow in a pipeline way and possibly span a number of routers.

3.2. Mixed-Criticality System (MCS)

Recently, Mixed-Criticality system have received broad attention.

Safety standards and industry practices have a different definition of MCS. Criticality relates to the functional safety of an application in industry practice and safety standards such as the IEC 61508, DO-178B and ISO 26262 standards[Ernst and Natale 2016]. In these industry standards sufficient independence or freedom from interference between functions of different criticality levels in both spatial and timing domains is required. The MCS model proposed in [Ernst and Natale 2016] answers to actual criticality requirements as defined by certification standards.

However, the definition of MCS first proposed by Vestal [Vestal 2007] and later supported by Burns and Davis [Burns and Davis 2017] specifies that the execution platforms must provide different operating modes for applications with different levels of criticality [Vestal 2007]. This definition makes the challenge of MCS is to achieve the necessary separation in order to ensure the time-constrained of high-critical applications, while providing efficient means of sharing resources in order to minimize the impact of sharing ressources on low-critical applications.

There are different objectives and different meanings assumed for some of the commonly used terms. and there are several papers that discuss this disconnect [Esper et al. 2015], [Graydon and Bate 2013], and [Paulitsch et al. 2015].

In this work, we have chosen to work with the Vestal model. According to this model, Mixed-Criticality Systems combine real-time software tasks with two or more distinct levels of criticality. In a MSC, we can find hard real-time applications, which have very stringent communication service requirements. It is absolutely necessary that all packets generated by a high-critical flow are delivered before or on their deadline even under the worst case scenarios.

We can also find soft real-time applications, which can tolerate some delays in the communication service. Messages generated by a soft real-time application will be called a low-critical flow in the sequel.

A MCS is defined to operate in a number of modes [Burns and Davis 2017]. A system starts operating in the normal mode, but a mode change may occur when at least one message is delayed for a longer time than is acceptable in the actual mode. If a system has more than one mode, then there must exist a mode change protocol to control how the system moves between modes [Burns 2014]. The system mode change is the feature allowing the system to move from one mode to another one. During system mode changes, flows may be suspended or aborted while flows that exist in both modes may have their WCCT changed.

A key aspect of MCS is that flow's parameters such as WCCT and/or period may be dependent on the criticality level of the system. Here, we consider the real-time network flow model proposed in [Shi 2009]. Each flow ρ_i has a set of properties and timing requirements such as O_i, T_i, $C_i(M)$ and $Crit_i$. The release time O_i is the first time a message of the flow ρ_i becomes ready to be transmitted. The period T_i is the fixed delay between releases of successive messages of the flow ρ_i. $C_i(M)$ is the maximum duration of transmission latency of a message, i.e., the WCCT. The WCCT $C_i(M)$ of the flow ρ_i depends on the current mode M. Here, we consider the maximum packet size belonging to the flow. We note that for two given modes, e.g. M_1 and M_2, if M_1 is more critical than M_2, then $C_i(M_1) \geq C_i(M_2)$ [Vestal 2007]. Finally, $Crit_i$ denotes the criticality of the flow ρ_i. In this work, we assume only two criticality levels: low-critical flows and high-critical flows.

We consider two modes: degraded mode and normal mode. The explanation of how those modes work and the mode change protocol is given in Section 5.

3.3. IF Language, IF Toolset, IFx Tool

IF language is used for real-time system specification. However, IF toolset provides a modeling and validation environment for asynchronous real-time systems. In this section, first, we present the IF language, then, we explain the validation approach using the IF toolset.

3.3.1. IF Language. A real-time system specification using IF language [Bozga et al. 2002; Bozga et al. 2004] is composed of active process instances running in parallel and interacting asynchronously through shared variables and messages (i.e., `signal` instances) passing via communication buffers (i.e., `signalroute` instances) or by direct addressing. IF processes describe sequential behaviors including communications (i.e., signal sending and receiving), process creation and destruction, and data transformations (e.g., variable assignments). These processes can be created and destroyed dynamically during the system execution.

A IF process is defined as a timed automaton extended with data, communication primitives, and urgency attributes on transitions (deadlines). Each IF process instance has a unique identifier number (i.e., instance `pid`), local data (clocks and discrete variables), control states, and a private input FIFO buffer storing the incoming messages. In IF Language, a transition is a process reaction in response to: (i) enabledness of provided expression (un-timed guard), (ii) enabledness of when constraint (timed guard), (iii) and the presence of input signal in the process instance buffer. Transition reaction is a set of actions (signal sending, process creation and destruction, variable assignments, clock setting, procedure calls, internal or observable action).

3.3.2. IF Toolset, IFx Tool. The IF toolset [Bozga et al. 2004; Bozga et al. 2002] provides a modeling and validation environment for asynchronous real-time systems. Figure 2 describes the architecture of the IF toolset which the core components are the syntactic transformations component and the exploration platform. From an IF specification, the first component allows the construction of an abstract syntax tree) which is a collection of C++ objects representing the syntactic elements present in the IF specification (e.g., processes, types, variables, etc.). In addition, this component has been used to construct code generators (e.g., simulation code), and static analysis transformations. For the exploration platform, his main features are: (i) process execution simulation (by using the abstract syntax trees generated by the first component), (ii) non-determinism resolution, (iii) management of time and representation of the state space (by composing all the active processes). This exploration platform can be connected to different test case generation and model-checking tools (e.g., TGV, CADP).

In the IF toolset, observers are used to specify and verify timing and liveness properties. These IF observers can control model generation process (by modeling and/or restricting the environment, cutting off generation of irrelevant states, cutting off execution irrelevant paths, etc.) and specify properties in an operational way.

IF language provides observer constructs for every parts of a system (e.g., states, variables), elapsed time and observable system events including input and output events, forking of processes, etc.

IF observers are described as an extended timed automaton which are executed in parallel with the target system (but have always the highest priority during state exploration). Observers can react synchronously to events and conditions occurring in the system. They are classified into: (i) `pure` observers to specify properties to be checked on the system), `cut` observers to cut selected executions paths), and `intrusive` observers to change the behavior of the system by changing variables or/and sending signals.

The IF/IFx tool[1] [2] is developed as an extended version of the IF toolset. IFx provides simulation features of the IF model and verification of properties such as deadlocks, timelocks, state invariants and properties expressed in observers or timing constraints.

In the following section, we describe the design of the proposed router.

[1]https://www.irit.fr/ifx/
[2]http://www-omega.imag.fr/tools/IFx/IFx.php

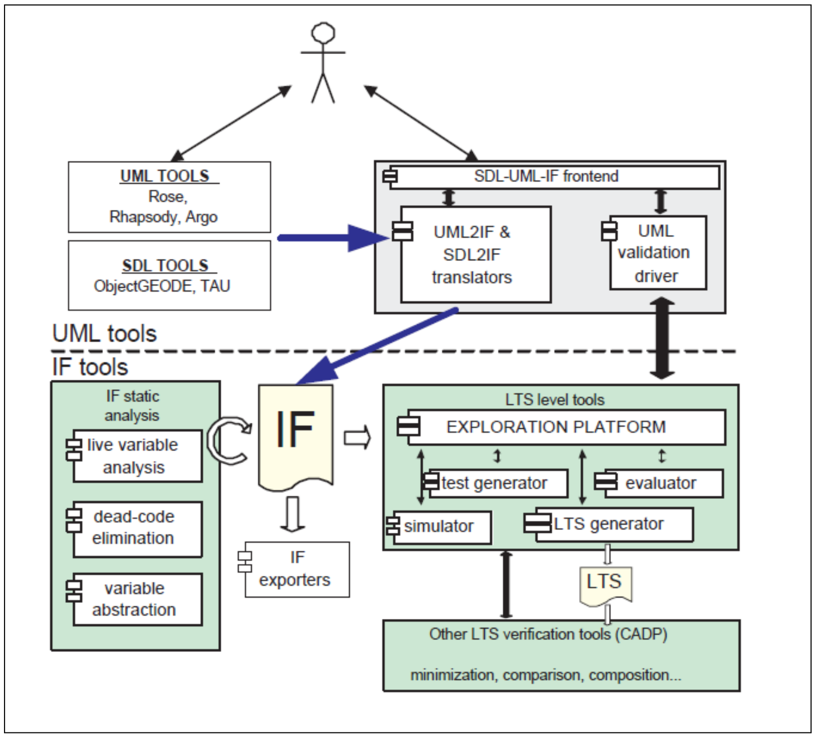


Fig. 2. The IF Toolset Architecture

4. THE PROPOSED ROUTER: DOUBLE ARBITER AND SWITCHING ROUTER (DAS)

The architecture of DAS is shown Fig. 3. N+1 virtual channels, input and output arbitration units, a routing logic, a VC allocator, a switching allocator and a crossbar constitute the router. The router combines 2 switching techniques: each port can use a Wormhole or a Store-And-Forward (SAF) switching technique depending on the criticality of the packets it deals with. In the sequel, we describe the use of the virtual channels and we explain why we use SAF for the high-critical flows and Wormhole for the low-critical flows. Then, we discuss how to set N and we detail the two stages of arbitration used in DAS.

4.1. N+1 Virtual Channels

The VCs 1 to N of DAS are dedicated to the high-critical flows and the VC N+1 is used for all low-critical flows. The N virtual channels of the high-critical flows use SAF switching. Each of this VC is dedicated to only one given high-critical flow. The last VC, dedicated to low-critical flows, is managed by a Wormhole switching technique. Notice that several low-critical flows can share this channel.

For example, if 3 high-critical flows share the same physical link, they need 3 VCs. Conversely, when 3 low-critical flows share the same physical link, they use the same virtual channel.

4.1.1. SAF for high-critical flows. High-critical flows are transmitted with a SAF policy and the preemption is managed at the packet level. In other words, a high-critical packet cannot be preempted by another flow. The main drawback of this policy is the input buffer cost that must be large enough to store the entire packet. We consider that this cost is limited in our context, since the analysis of real-life critical systems shows

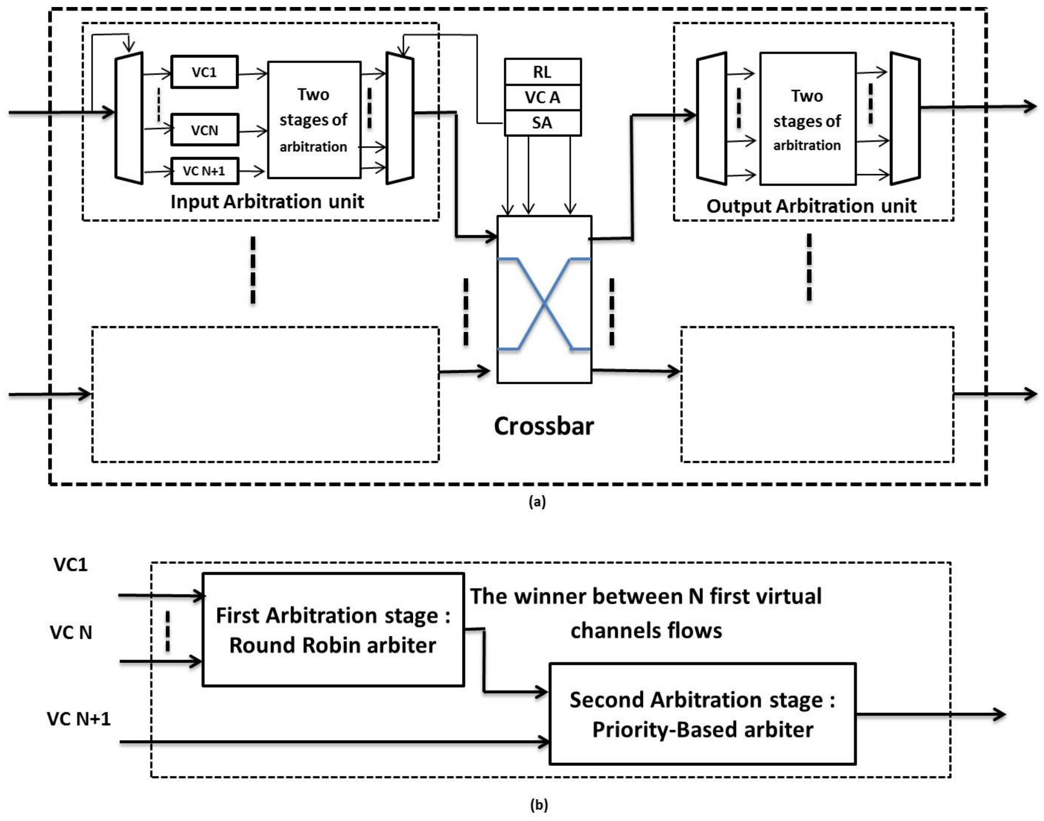

VC1
VCN
VC N+1
Two stages of arbitration
Input Arbitration unit
RL
VC A
SA
Crossbar
Two stages of arbitration
Output Arbitration unit
(a)
VC1
VC N
VC N+1
First Arbitration stage : Round Robin arbiter
The winner between N first virtual channels flows
Second Arbitration stage : Priority-Based arbiter
(b)

Fig. 3. DAS architecture: (a) Architecture, (b) Stages of arbitration RL stands for Routing Logic, VC A for the Virtual Channel Allocator and SA for Switch Allocator

that high-critical flows are usually characterized by small packets size (e.g. sensor and control signals). For instance, in the avionics Rosace benchmark [Pagetti et al. 2014], it does not exceed 3 flits of 32 bits. The payload size of controller area network (CAN) which is a widely-used bus protocol in automotive distributed embedded systems, does not exceed 8 bytes [Andrade et al. 2018]. Finally, in the ARINC429 protocol which is an open standard and one of the most adopted protocol in the aeronautical industry, the payload size does not exceed 24 bits [Santos and d'Amore 2018].

In a Wormhole policy, a packet can be stored over multiple routers and then can occupy several physical links simultaneously. Consequently, it increases the potential congestion over the network. The links used by a packet are unavailable for other packets arriving into the router, and this additional blocking time makes difficult the computation of the WCCT [Indrusiak et al. 2016]. On the contrary, by using a SAF policy, each packet is allocated to only one link at a time and the congestion can be controlled without a prohibitive cost considering small high-critical packets.

From the computation of the WCCT for each flow, we can decide about the schedulability of flows. If the WCCT of a flow is smaller or equal than its deadline, the flow is said to be schedulable. We consider that the flow set is schedulable when all the flows are schedulable.

SAF and packet-level preemption allow us to minimize the pessimistic degree of the computed WCCT. Moreover, by allocating one virtual channel to each high-critical flow, we can adapt the real-time scheduling analysis proposed in [Shi and Burns 2008] to SAF policy and compute offline the WCCT for high-critical flows (see section 5.2). As a result, SAF switching is intended to improve the schedulability of flows.

4.1.2. Wormhole policy for low-critical flows and Flit-level Preemption. Low-critical flows are transmitted with a Wormhole policy. Wormhole is largely adopted in NoCs because it does not require large capacity buffers and, at the same time, it minimizes the communication time.

In order to ensure predictable communication time and minimal interference delays for high-critical flows, we need to preempt a low-critical flow as soon as possible when a high-critical one occurs. So, the preemption is implemented at the flit level for the last VC. In other words, high-critical flows, which use the N first VCs, can preempt any low-critical flows at flit level.

4.1.3. Number of virtual channels. The number N of VCs allocated to high-critical flows depends on communication requirements of the software tasks, but also of the mapping of these tasks. On the one hand, for a fixed task mapping, we can choose N as the maximum of high-critical flows sharing the same link. We note that the higher the value of N, the larger the overhead of area for the NoC implementation. On the other hand, for a fixed N, we must choose a task mapping which allows us to have at most N high-critical flows sharing the same physical link. This kind of problems is similar to a Quadratic Assignment Problem (QAP) [Al Faruque and Henkel 2008], which is known as an NP-Hard optimization problem. There are however several heuristics that can be used for defining such mapping as those described in [Al Faruque and Henkel 2007] or [Al Faruque and Henkel 2008]. These articles present multi-criteria heuristics to optimize the total communication volume and the number of VCs. This optimization issue is orthogonal to the scope of this article.

4.2. The Two Stages of Arbitration

At each cycle, in DAS, only one virtual channel can advance from an input port, and only one virtual channel can be accepted by each output port. DAS implements input and output arbitration units in order to solve these problems. As shown in Fig. 3(b), the input and output arbitration units used in DAS are based on a combination of fair and priority-based algorithms. In the sequel, we describe these units.

4.2.1. Input Arbitration Unit. Many VCs of the same input port can ask to advance to different or the same output port while the router can accept just one VC from each input port at each cycle. The main task of input arbitration unit is to choose one virtual channel for each input port.

4.2.2. Output Arbitration Unit. Many VCs of different input ports can ask to advance to the same output port while only one VC can be accepted. The main task of output arbitration unit is to choose one candidate VC for each output port.

Input and output arbitration units are based on 2 stages of arbitration. The first stage is a round robin arbitration between the N first VCs while the second stage is a priority-based arbitration between the winner of the first stage and the last VC (VC N+1).

The first stage has to be fair and gives equal chance between all the high-critical flows, while the second stage provides the flit-level preemption to high-critical flows. So, the winner of the first stage can preempt at flit-level the last VC used by low-critical flows.

In the following section, we present the different modes of DAS and we explain how and when the system moves from one mode to another.

5. SYSTEM MODE CHANGES

We consider two modes: degraded mode and normal mode. In normal mode, all flows can meet their deadlines. However, in degraded mode, only the high-critical flows meet their deadlines, while some low-critical flows will miss theirs. We notice that in degraded mode, only high-critical flows are routed.

Each I/O port of the network's router may be either in degraded mode or in normal mode. We cannot assign a mode per router because each of its I/O port has its own status. For example, we can have many conflicts on one port while other ports of the same router are available.

In the sequel, we explain how and why the system moves from one mode to another one. Then, we present the related WCCT analysis model. Finally, we discuss our choice of SAF switching mode for MCS.

5.1. Principle of Mode Change

In normal mode, an input or an output port is used by all flows, without interference, whatever their respected criticality is. On the contrary, in degraded mode, only high-critical flows transit through the port.

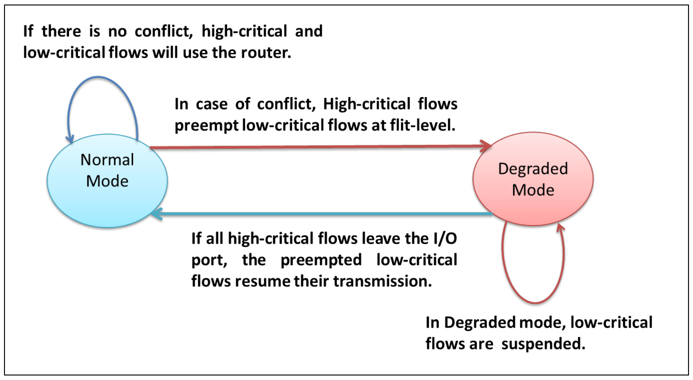

If there is no conflict, high-critical and low-critical flows will use the router.
In case of conflict, High-critical flows preempt low-critical flows at flit-level.
Normal Mode
Degraded Mode
If all high-critical flows leave the I/O port, the preempted low-critical flows resume their transmission.
In Degraded mode, low-critical flows are suspended.

Fig. 4. System mode change for I/O port

The system mode change depends on the result of the router arbiters. Fig. 4 summarizes it by a transition graph. Two stages of arbitration manage the mode change of each I/O port. The mode change behavior is intended to respect the four properties stated below. Those properties are verified with Model-Checking using IFx toolset in the Section 6.

Property 1. In normal mode, high-critical and low-critical flows use the router without having conflict.
Property 2. In degraded mode, low-critical flows are suspended.
Property 3. High-critical flows always preempt low-critical flows at flit-level.
Property 4. Preempted low-critical flows resume their transmission after the end of transmission of high-critical flows.

Considering Property 1, In normal mode, high-critical and low-critical flows share the same router with set time spacing. Considering Property 3, If a high-critical flow asks for an input already used by a low-critical flow, the related input port changes its mode into the degraded mode. Similarly, if a high-critical flow asks for an output already used by a low-critical flow, the related output port changes its mode into degraded mode. In other words, if an arbiter detects an interference between flows of different level of criticality on an input and/or an output port, the I/O port switches to degraded mode in order to ensure the timing constraints for high-critical flows.

Considering Property 2 and 4, the high-critical flows preempt, at the flit level, the low-critical flows. When all high-critical flows have left the correspondent input and output ports, the I/O port returns into the normal mode and is allowed for receiving low-critical flows again.

Operating the previous modes leads to a deterministic and accurate WCCT. In the sequel, we present the WCCT analysis model for high-critical and low-critical flows for each mode.

5.2. Worst case communication time analysis

In this paragraph, we present the WCCT analysis of high-critical and low-critical flows for DAS.

5.2.1. High-critical flows. The WCCT $C_i(M)$ of a high-critical flow ρ_i depends on mode M of each used I/O port. We assume that $C_i(M)$ is equal to the sum of unitary worst case communication time for all routers belonging to its communication path:

$$C_i(M) = C_i(S_0, ..., S_n) = \sum_{j=0}^{n-1} cu_{i_j}(S_j) \tag{1}$$

where $C_i(M)$ is the WCCT of a flow ρ_i with n hops. M is a vector of the modes of each I/O port used along the path of the flow. $cu_{i_j}(S_j)$ is the unitary worst case communication time for the j^{th} hop of a flow ρ_i, i.e., the maximum duration of transmission latency for a given hop. It is defined as the time spent by the router to send the message to the next router. It depends on the mode of the used input and output ports. S_j represents the combinations of the input and output port modes for the j^{th} hop of a flow ρ_i (see Table II).

We now investigate the unitary worst case communication time cu_{i_j} for each I/O port mode. The WCCT may be composed of the path delay (PD), the direct interference delay (DID) [Dridi et al. 2016] and the preemption delay (PTD).

Let take the following example in order to explain Eq(1). We consider one high-critical flow, ρ_1, as shown in Fig. 5.

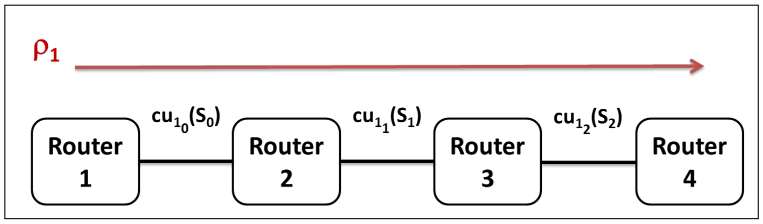


Fig. 5. The unitary worst case communication time

ρ_1 uses two physical links, then:

$$C_1(M) = C_1(S_0, ..., S_1) = \sum_{j=0}^{1} cu_{1_j}(S_j) = cu_{1_0}(S_0) + cu_{1_1}(S_1)$$

Definition 5.1 (***Path Delay, PD***). A path delay, noted PD, is the delay required to send the packet from one router to the next router when no traffic flow contention exists.

The packet size and the link bandwidth mainly determine PD, and then:

$$PD = size_{max}/B_{link} + S \tag{2}$$

where S represents a constant processing delay in each router, B_{link} is the link bandwidth of the link between 2 routers, and $size_{max}$ is the maximum packet size belonging to the flow.

Let see now direct interference delays.

Definition 5.2 (***Direct Interference Delay, DID***). A direct interference delay, noted DID, occurs when 2 flows want to access at the same time to a given physical link.

We assume the packet from the observed traffic-flow is sent just after a high-critical packet which uses the same physical link (same I/O port).

$$DID = number_{flow} \cdot PD \tag{3}$$

where $number_{flow}$ indicates the number of high-critical flows which share the same physical link.

Now, let take the following example in order to explain Eq(3). We consider 3 flows, ρ_1, ρ_2 and ρ_3, as shown in Fig. 6. Here, we focus on the DID of the flow ρ_1.

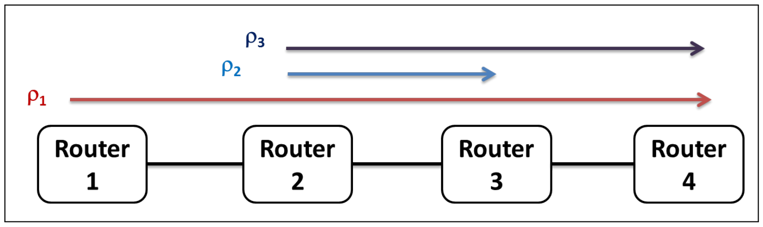


Fig. 6. Direct interference delay

The first physical link is used only by ρ_1. Then DID of ρ_1, for the first hop, is:

$$DID = number_{flow} \cdot PD = 0 \cdot PD = 0$$

The second physical link is used by ρ_1, ρ_2 and ρ_3. Then DID of ρ_1, for the second hop, is:

$$DID = number_{flow} \cdot PD = 3 \cdot PD$$

The third physical link is used by ρ_1 and ρ_2. Then DID of ρ_1, for the third hop, is:

$$DID = number_{flow} \cdot PD = 1 \cdot PD$$

The last delay composing the WCCT is the preemption delay.

Definition 5.3 (***PreempTion Delay, PTD***). The preemption delay is the maximum blocking delay due to any low-critical flow which has already begun transmission.

The maximum blocking delay happens when a high-critical packet arrives just after a low-critical packet has started its service.

Considering our flit-level preemption provided by DAS, for each hop, high-critical flow waits for at most one flit delay and then starts its transmission.

The preemption time is computed by:

$$PTD = f_{size}/B_{link} \tag{4}$$

Where the flit size is f_{size} and the link bandwidth is B_{link}. As both f_{size} and B_{link} are constant, PTD can be considered as constant too.

In a wormhole NoC, we also have indirect interference delays. There is an indirect interference when two flows do not share any physical link but the have direct interference with the same flows [Shi 2009]. This kind of interference turns challenging the communication time analysis [Dridi et al. 2016]. The most pessimistic part in existing WCCT analysis for NoCs architectures is indirect interference. The more we have indirect interference, the more pessimistic WCCT analysis we will have.

Using Wormhole switching mode, the packet is divided into several flits. In each cycle the packet occupies different routers, which increase the probability to have contention with other flows [Shi 2009]. On the contrary, using a SAF switching mode, the packet occupies only one router, which decreases the probability to have contention with other flows. Combining SAF with virtual channel avoid indirect interference latency. Consequently, the WCCT analysis is more accurate. This is why apply this approach with DAS.

Table II. Worst Case Communication Time of high-critical flows

S	**Input port mode**	**Output port mode**	**Unitary worst case communication time** cu_{i_j}
normal	normal	normal	PD + DID (1)
degraded	degraded	normal	PD + DID + PTD (2)
degraded	normal	degraded	PD + DID + PTD
degraded	degraded	degraded	PD + DID + PTD

The Table II presents the unitary worst case communication time, in different modes, for the j^{th} hop of the high-critical flow ρ_i. As shown in Table II, we add preemption delay when at least one of the I/O port switches to degraded mode.

In MCS, the scheduling analysis is done for each mode. In normal mode, high-critical and low-critical flows are scheduled without interference. Therefore, we consider for high-critical flows the lowest pessimistic unitary Worst Case Communication Time given by the expression (1) in Table II.

In degraded mode, interferences between high and low-critical flows have been detected, but only high-critical flows have to meet their deadlines in that mode. So, we consider the expression (2) in Table II which takes into account the preemption delays due to low-critical flows.

5.2.2. Low-critical flows. For the low-critical flows, we use an analysis method by Shi et al. described in [Shi and Burns 2009].

In order to analyze the communication time of low-critical flows with a priority share policy deployed over Wormhole NoC architecture, Shi et al. introduces 2 kinds of priority levels: natural priority and system priority. Those 2 priority levels are assigned off-line and remain constant at run-time.

The natural priority is produced relying on a distinct priority per flow policy. Flows which use the same virtual channel will be mapped to the same system priority.

For system and natural priority, the value 1 denotes the highest priority and larger integers denote lower priorities. In our case, the system priority of all low-critical flows is 2, while the system priority of high-critical flows is 1. The natural priority of all low-critical flows is the same because DAS provides one virtual channel to low-critical flows.

Considering natural and system priorities of each flow, [Shi and Burns 2009] assumed that delays of wormhole communications with share policy are composed of the interferences from higher priority flows and the blocking from flows with the same system priority. Based on different priority levels and the competing relationships, they categorize the delays into four different types:

(1) Direct interference from flows with the higher system priority;
(2) Indirect interference from flow with the higher system priority;
(3) Direct blocking from flows with the same system priority;
(4) Indirect blocking from flows with the same system priority.

Table III. Flow Model

Flow	Criticality	Size	Source	Destination	Period	Deadline
ρ_1	High-critical	2 flits	1	4	10	10
ρ_2	High-critical	2 flits	2	3	10	10
ρ_3	Low-critical	8 flits	2	4	10	10

5.2.3. Example. Now, we take again the previous example in order to explain the given WCCT analysis. Table III presents the flow model, i.e., the attributes of each flow. We consider 3 flows as shown in Fig. 6. ρ_1 and ρ_2 are high-critical flows while ρ_3 is a low-critical flow. We assume that all flows use the same units of time, and then, units of time are not specified here.

In this example, we consider that $B_{link} = 1$ and $PTD = 1$.

We summarize in the sequel how the WCCT of each flow is computed.

WCCT of ρ_1. WCCT for this flow is computed as follow:

$$C_1(M) = C_1(S_0, ..., S_2) = \sum_{j=0}^{2} cu_{1_j}(S_j) = cu_{1_0}(S_0) + cu_{1_1}(S_1) + cu_{1_2}(S_2)$$

with:

$$cu_{1_0}(normal) = PD + DID = 2/1 + 0 = 2$$
$$cu_{1_0}(degraded) = PD + DID + PTD = 2/1 + 0 + 0 = 2$$

We notice that $PD = 0$ and $PTD = 0$ because the physical link between the routers 1 and 2 is only used by ρ_1.

$$cu_{1_1}(normal) = PD + DID = 2/1 + 1 \cdot 2 = 4$$
$$cu_{1_1}(degraded) = PD + DID + PTD = 2/1 + 1 \cdot 2 + 1 = 5$$

$$cu_{1_2}(normal) = PD + DID = 2/1 + 0 = 2$$
$$cu_{1_2}(degraded) = PD + DID + PTD = 2/1 + 0 + 1 = 3$$

Here, $PD = 0$ since ρ_1 is the only high-critical flow who uses the physical link between the routers 3 and 4.

WCCT of ρ_2. WCCT for this flow is computed as follow:

$$C_2(M) = C_2(S_0) = cu_{2_0}(S_0)$$

$$cu_{2_0}(normal) = PD + DID = 2/1 + 1 \cdot 2 = 4$$
$$cu_{2_0}(degraded) = PD + DID + PTD = 2/1 + 1 \cdot 2 + 1 = 5$$

WCCT of ρ_3 . ρ_3 is a low-critical flow. So, we apply the WCCT analysis given in [Shi and Burns 2009].

5.3. Comparison between SAF and Wormhole for MCS

In this section, we explain our choice of SAF switching mode and how it is more efficient than wormhole switching mode for MCS. There is two motivations for such choice. First, SAF provides the minimum WCCT comparing to other switching modes. Second, the WCCT analysis of SAF is the less pessimistic solution comparing to other switching modes.

5.3.1. Minimizing WCCT. In most cases, Wormhole switching mode provides the minimum WCCT comparing to other switching modes, but considering the assumptions of this work, Wormhole loses its effectiveness against SAF switching mode.

Combining virtual channel with SAF allows us to avoid indirect interference. Therefore, the WCCT of SAF is composed only by the path delay (PD), the direct interference delay (DID) and the preemption delay (PTD).

However, the WCCT of wormhole is composed by the path delay (PD), the direct interference delay (DID), the indirect interference delay (IID) and the preemption delay (PTD) [Dridi et al. 2016]. The WCCT of wormhole is then given by:

$$WCCT_{wormhole} = PD_{wormhole} + DI_{wormhole} + IID_{wormhole} + PTD$$

In all cases, we note that $PD_{wormhole}$ is less than PD_{SAF}. The difference between $PD_{wormhole} and PD_{SAF}$ is decreased since messages of high critical flows are small.

In addition, for high throughput in the network, the indirect interference delay will be significantly increased in the WCCT anlysis of Wormhole. However, the WCCT analysis of SAF based router do not suffer from indirect interference delay even for high throughput in the network.

To conclude, wormhole losses most of its effectiveness against SAF switching mode considering small message and high throughput.

In section 6.1, we evaluate the communication time of DAS and a Wormhole NoC for different messages sizes and different throughput levels.

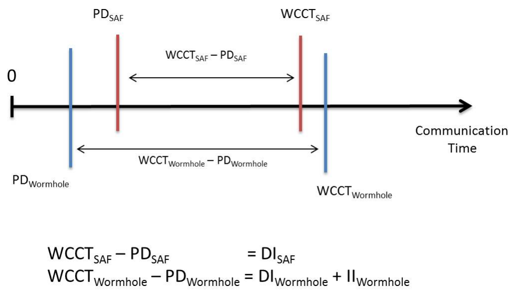

PD_SAF
WCCT_SAF
WCCT_SAF – PD_SAF
0
Communication Time
WCCT_Wormhole – PD_Wormhole
PD_Wormhole
WCCT_Wormhole
WCCT_SAF – PD_SAF = DI_SAF
WCCT_Wormhole – PD_Wormhole = DI_Wormhole + II_Wormhole

Fig. 7. Communication Time of SAF and Wormhole NoC

5.3.2. Less pessimistic solution. The fundamental issue with MCS is how to enforce resource guarantee for high-critical flows and efficient resource utilization for low-critical flows.

In the context of NOC architectures, we want to ensure timing constraints of high-critical flows and to minimize the impact of shared resources on low-critical flows (i.e. increase the throughput of low-critical flows). The temporal validation of a real-time system relies on a set of worst-case behaviors depending on the task and communication model. Then, for schedulability analysis, we do not look for solutions which only minimize communication times but we look for solutions which provide a reasonable communication time with a less pessimistic worst case communication time analysis.

Fig 7 compares the degree of pessimism using the two solutions, assuming small size message with high throughput. The degree of pessimism is the difference between the path delay and the worst case communication time. As shown in Fig 7, $PD_{wormhole}$ is less than PD_{SAF}. For high throughput in the network, the WCCT of SAF may be less than WCCT of wormhole because of indirect interference.

Assuming small size messages for high-critical flows and high throughput, Wormhole presents a pessimistic schedulability analysis comparing with SAF.

To conclude, SAF is more suitable than Wormhole switching mode for MCS. It is not only providing the minimum WCCT, but it presents also the less pessimistic WCCT analysis comparing with wormhole. In the sequel, we present multi-abstraction level evaluations of DAS.

6. IMPLEMENTATION AND MULTI-ABSTRACTION LEVEL EVALUATIONS

In the previous section, we proposed DAS, a router for MCS. DAS enforces high-critical flow temporal constraints and maximizes low-critical flow throughput. We also formalize the WCCT of low-critical and high-critical flows. WCCT formalization models the temporal behavior of DAS.

In this section, we explain how we verify DAS behavior. We evaluate how DAS fulfil the MCS requirements. The evaluation is made at three levels of abstraction.

At the higher level of abstraction, we choose Model-Checking to validate the arbitration policies and the mode changes implemented in the router. The Section 6.5 gives some details about this approach.

In Section 6.1, we use simulation techniques that handle an intermediate abstraction level, in order to measure the communication time through the DAS router, and then the efficiency of the system mode change for reducing the high-critical flow WCCT.

In addition, we evaluate the impact of communication delays on tasks. For such a purpose, we use a longitudinal flight controller benchmark called Rosace [Pagetti et al. 2014].

Simulations are based on a transaction-Level modeling (TLM) of routers in SystemC.

At a lowest level of abstraction, an hardware synthesis allows us to verify the realism of DAS architecture from the circuit area point of view. The synthesis is done from a Verilog-HDL model of the router (see Section 6.4).

6.1. Evaluation of communication latency with SystemC simulations

In this subsection, we evaluate the impact of resource sharing on high-critical flows in order to check the ability of DAS to bound communication delays. Next, we measure the additional latency on low-critical flows due to the resource reservation for high-critical flows.

To perform these evaluations, we have implemented DAS in the cycle accurate SystemC-TLM simulator SHOC [Seplveda et al. 2009]. SHOC provides all NoC components for the simulation of many-core architectures. It also supports different types of traffic generators and consumers, and allows us to observe the traffic in the NoC.

The simulation results are established for 3 different NoCs. The first one is DAS, the router we proposed in Section 4. The second one is based on a classical architecture of wormhole virtual channel router (VC router). The third one is a wormhole NoC router which uses flit-level priority preemption (WNoC). All routers have the same dimension and topology, i.e., 4*4 2D-mesh. They use the same XY routing algorithm. They use 5 virtual channels per port.

First, we compare DAS with VC router using uniform pattern traffic. Then, we compare DAS with WNoC using All-To-One pattern traffic.

6.1.1. DAS Vs VC router with uniform pattern traffic: Evaluation of high-critical flow latency . In this section, we verify the ability of DAS to bound the communication delays of high-critical flows by simulations.

For all simulations, one high-critical flow is assigned to a randomly generated source and destination node. Then, we perform 100 SHOC simulations by increasing the use rate of the network. For each simulation, we generate low-critical flow sets which share some physical links with the high-critical flow. For this experiment, the size of the high-critical flow is 2 flits while the size of a low-critical flow is 8 flits. The period and the release time of each flow are randomly generated. The generation of each flow set is done by applying UUniFast [Bini and Buttazzo 2005].

Fig. 8 and Fig. 9 show the latency of the high-critical flow with different use rates of the network; The high-critical flow goes respectively through 4 and 3 physical links from its source to its destination node.

Fig. 8 and Fig. 9 show that DAS is able to reduce significantly the latency of high-critical flows because of the combination between SAF and virtual channels with flit-level preemption. For 15% of network use rate and comparing to a VC router, DAS reduces by 80% the additional latency for a high-critical flow using 3 links.

To conclude, using DAS, high-critical flows are less affected by the sharing of resources with low-critical flows.

6.1.2. DAS Vs VC router with uniform pattern traffic: Evaluation of low-critical flow latency. In this section, we quantify the latency added by DAS for low-critical flows due to high-critical flows resource reservation comparing to virtual channel routers.

In contrast to the previous evaluation, for each measure in this experiment, one low-critical flow is assigned to a randomly generated source and destination node. In order to increase the network use rate, we perform 100 SHOC simulations by increasing the number of high-critical flows and by decreasing the period of high-critical flow. We note that the generation of each flow set is done by applying UUniFast [Bini and Buttazzo 2005]. Then, we measure the low-critical flow latency. The generated high-critical flow set shares the same physical links with the low-critical flows. For this experiment, the size of a high-critical flow is 2 flits while the size of a low-critical flow is 8 flits. The release times of each flow are also randomly generated.

For different use rates of the network, Fig. 10 shows the latency of low-critical flows. Results show that, compared to a virtual channel router, some additional delays for low-criticality flows are introduced by the system mode change and the preemption of low-critical flows used by DAS. For 15% of network use rate and comparing to a VC router, DAS increases by 25% the additional latency for a low-critical flow using 3 links.

Notice that in MCS, low-critical flows can tolerate some additional delays without damaging the integrity of the whole system.

6.1.3. DAS vs WNoC with All-To-One pattern traffic: Evaluation of high-critical flow latency. In this section, we verify the ability of DAS to minimize the communication delays of high-critical flows by simulations comparing with WNoC.

For all simulations, one high-critical flow is assigned to a randomly generated source and destination node. We called this flow the first high-criitical flow. Then, we perform 100 SHOC simulations by increasing the use rate of the network. In order to cereate contention between the first high-critical low and other high-critical flows, we generate,for each simulation, a high critical flow set of 10 flows which share some physical links with the first high-critical flow. In order to cereate contention between the first high-critical flow and low-critical flows, we generate,for each simulation, a low critical flow set of 40 flows which share some physical links with the first high-critical flow. All-To-One traffic pattern is used. in All-To-One traffic pattern, the source node will

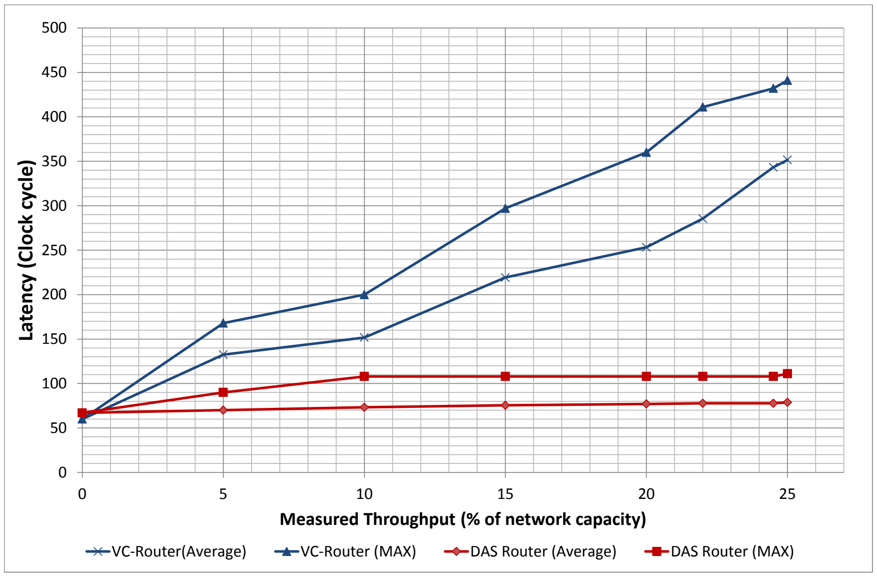


Fig. 8. Latency of high-critical flow with 3 physical links

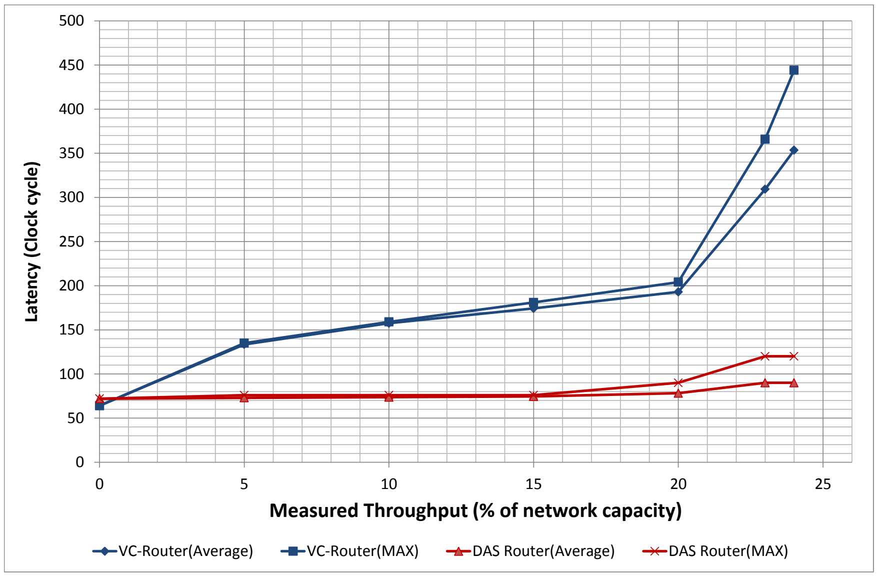


Fig. 9. Latency of high-critical flow with 4 physical links

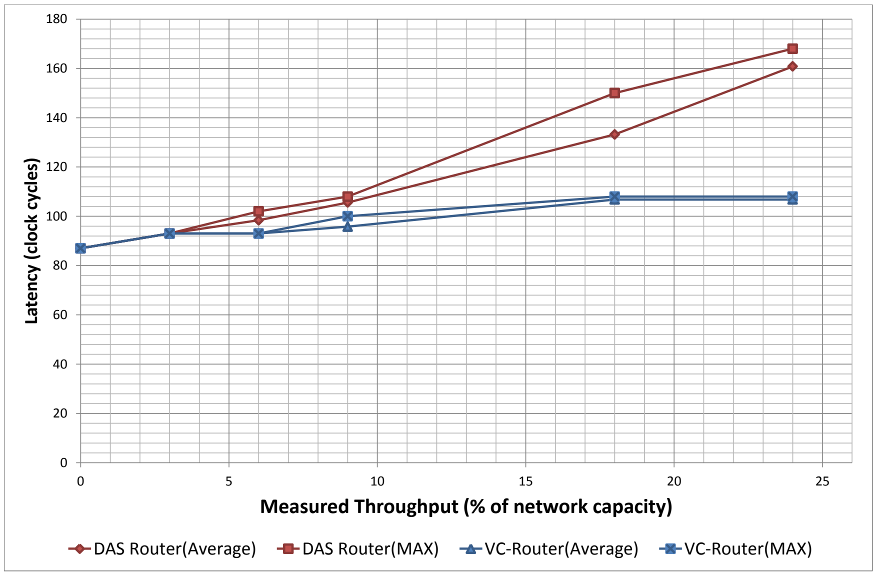


Fig. 10. Latency of low-critical flows with 4 physical links

be selected randomly using UUniFast [Bini and Buttazzo 2005], while the destination node is already fixed by the user. All flows of the same set have the same destination node. For his experiment, the size of the high-critical flow is ranging from 2 to 6 flits with step of 2 flits while the size of a low-critical flow is 8 flits. The period and the release time of each flow are randomly generated.

Fig. 11 shows the latency of the first high-critical flow with different use rates of the network using DAS and WNoC. Fig. 11(a) shows the latency of the high-critical flow of 2 flits with different use rates of the network and with path distance of 4 and 3 physical links. Results show that DAS is able to reduce significantly the latency of high-critical flows comparing with WNoC because the combination between SAF and virtual channels with flit-level preemption and the impact of indirect interference on communication latency for WNoC. For 14% of network use rate and comparing to a WNoC, DAS reduces by 64% the additional latency for a high-critical flow using 4 links.

Fig. 11(b) shows the latency of the high-critical flow of 4 flits with different use rates of the network and with path distance of 4 and 3 physical links. The comparison between WNoC and DAS for messages of 4 flits shows similar result with a small advantage for DAS in high throughput.

Fig. 11(c) shows the latency of the high-critical flow of 6 flits with different use rates of the network and with path distance of 4 and 3 physical links. Results show that DAS losses its effectiveness against WNoC.

To conclude, considering small packets (no more than 4 flits) and high throughput, DAS is more efficient than WNoC which makes DAS able to satisfy our MCS requirements. However, considering packet with more than 6 flits, DAS losses its effectiveness against WNoC which make Wormhole the most interesting switching mode for hard real-time applications deployed over NoC architectures today.

6.1.4. DAS vs WNoC with All-To-One traffic pattern: Evaluation of low-critical flow latency. In this section, we quantify the latency added by DAS for low-critical flows due to high-critical flows resource reservation comparing to WNoC.

In this experiment, one low-critical flow is assigned to a randomly generated source and destination node. Then, we generate a set of high-critical flow which share the same physical links with the low critical flow. We note that the used traffic pattern of high critical flows is All-To-One. We perform 100 SHOC simulations by increasing the number of high-critical flows and by decreasing the period of high-critical flows in order to increase the network use rate. For this experiment, the size of a high-critical flow is 2 flits while the size of a low-critical flow is 8 flits. The release times of each flow are also randomly generated using UUniFast [Bini and Buttazzo 2005].

Fig. 12 shows the latency of low-critical flows, for different use rates of the network. Results show that, compared to a WNoC, some additional delays for low-criticality flows are introduced by the preemption of low-critical flows and SAF switching mode used by DAS for high-critical flow. For 8% of network use rate and comparing to a WNoC, DAS increases by 21% the additional latency for a low-critical flow using 3 links.

In MCS, the first requirement is to ensure the timing constraints of high-critical flow. Providing a fast communication for low-critical flow without ensuring timing constrained of high-critical flow do not answer to the MCS requirements.

To conclude, DAS presents the best communication time for high-critical flow comparing with WNoC and VC router since messages of high critical flows are small. Comparing with other routers such as WNoC and VC router, DAS does not provide the best communication time of low-critical flows. In this section, we quantify the communication time for low and high-critical flows using DAS. In the sequel, we study the impact of communication delay on applications using DAS and WNoC.

6.2. Case study based on the Rosace benchmark

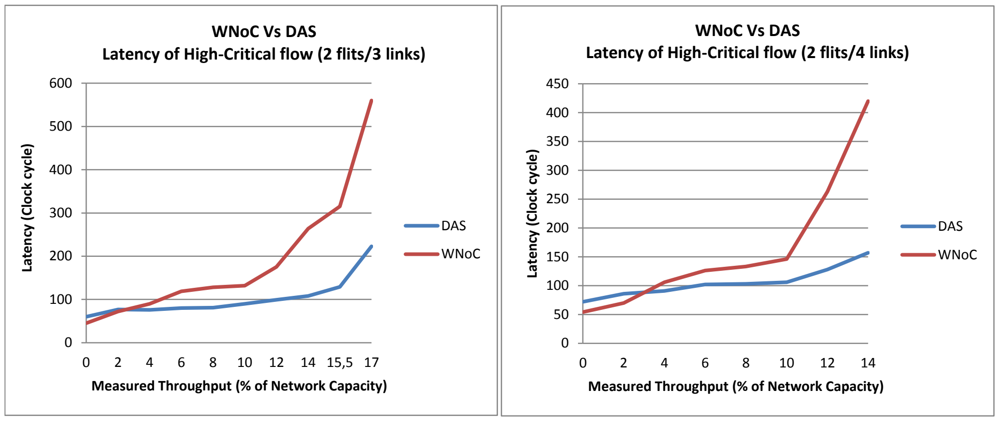
In this section, we evaluate the impact of communication delays on a case study. We use a well known benchmark in critical system called Rosace. Rosace is a longitudinal flight controller [Pagetti et al. 2014].

The simulation results are established for DAS and WNoC. Both routers have the same dimension and topology, i.e., 4*4 2D-mesh. They use the same XY routing algorithm. and they use 5 virtual channels per port.

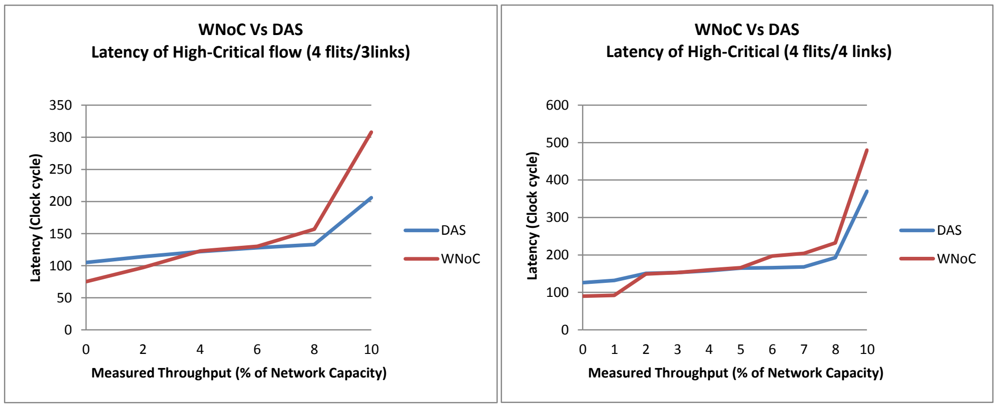
First we describe the task model, the task mapping and the flow model of the case study. Then, we evaluate the communication delays of DAS and WNoC on Rosace task deadlines.

6.2.1. Rosace: a Longitudinal Flight Controller. Rosace is a flight controller software of medium-range civil aircraft in en-route phase. The autopilot of this software commands a constant vertical speed Vz to change the cruise level until capturing the new flight level. The autopilot maintains a constant altitude h and the autothrottle maintains the airspeed Va during the cruise subphase. The goal of this application is to track all along the flight: the altitude, the vertical speed and the airspeed commands (respectively h, Vz and Va).

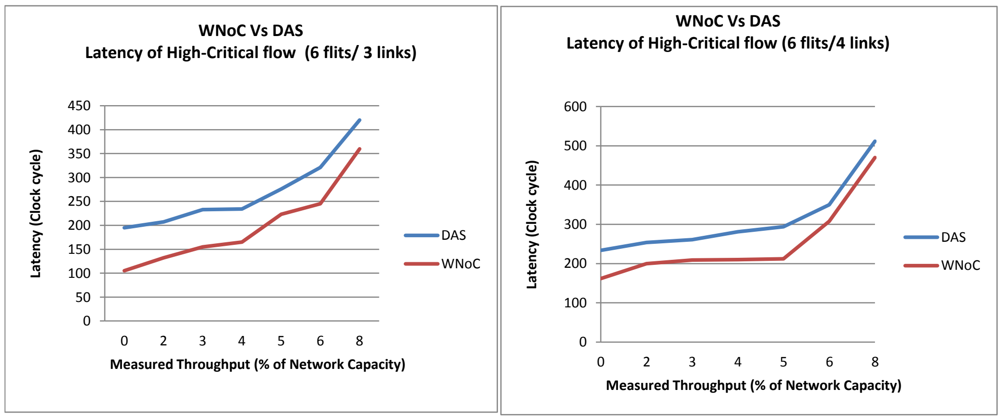
This flight controller is a multi-periodic real-time C application. Fig 13 shows the task graph of this controller. The controller is composed of two control loops. The control loop composed of the tasks $\tau 1$, $\tau 2$, $\tau 3$, $\tau 4$, $\tau 5$, $\tau 7$, $\tau 8$, $\tau 9$, is the altitude control loop, which maintains and tracks the vertical speed Vz. Tasks $\tau 1$, $\tau 4$, $\tau 5$, $\tau 6$, $\tau 10$, $\tau 11$



(a)



(b)



(c)

Fig. 11. Latency of high-critical flow with DAS and WNoC
(a) size = 2 flits (b) size = 4flits (c) size = 6 flits

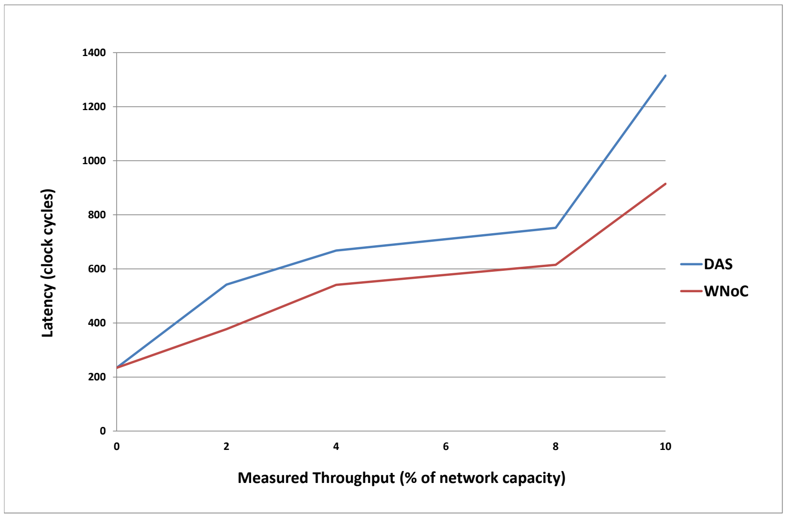


Fig. 12. Latency of low-critical flows

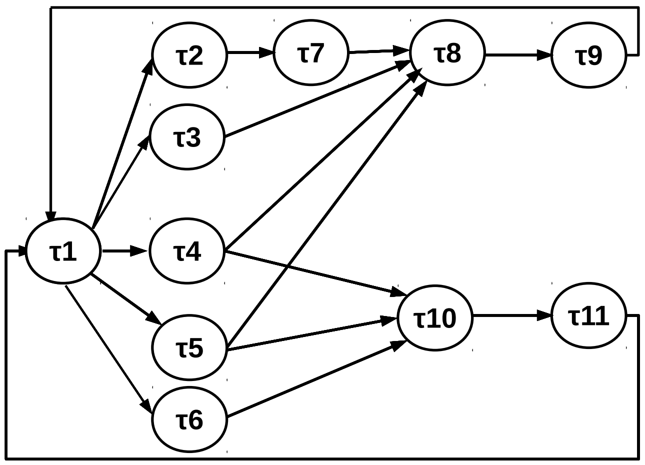


Fig. 13. Rosace Task Graph

implement the airspeed control loop that maintains and tracks the desired airspeed Va.

Each task τ_i is periodic and defined by the following timing parameters (R_i, T_i, $WCET_i$, D_i, $Node_i$). R_i is the release time of the task. T_i is the task period. $WCET_i$ is the task worst case execution time. D_i is the deadline of the task. $Node_i$ identifies the node which executes the task. This parameter allows us to introduce the task mapping on the NoC. Table IV presents the task model and the task mapping. For the task mapping, we choose a task mapping which map each critical task in non-used processor in order to create the maximum of communication over the network.

Rosace is a critical real-time application and all its tasks require to strictly respect their deadlines.

Table IV. Task Model

Task	R_i	T_i (μs)	$WCET_i$ (μs)	D_i (μs)	$Node_i$
τ_1	0	5000	200	5000	1
τ_2	0	10000	100	10000	6
τ_3	0	10000	100	10000	8
τ_4	0	10000	100	10000	2
τ_5	0	10000	100	10000	0
τ_6	0	10000	100	10000	4
τ_7	0	20000	100	20000	3
τ_8	0	20000	100	20000	5
τ_9	0	20000	100	20000	7
τ_{10}	0	5000	100	5000	13
τ_{11}	0	5000	100	5000	14

6.2.2. Evaluation of the communication time impact on the Rosace application. Previous experiments evaluate the flow latencies for different network workloads and different traffic patterns. Now, we evaluate the communication time impact on the Rosace application using DAS and WNoC.

In order to perform our simulation, we use DTFM [Dridi et al. 2016]. DTFM allow us to compute the flow model from the task model, the NoC model and the task mapping. For Rosace, the obtained flow model is used to feed SHOC (DAS and WNoC).

After computation of the Rosace flow communication model, we perform 100 SHOC simulations by increasing the use rate of the network. For each simulation, we generate 50 low-critical flow set which share some physical links with the Rosace communication model. All-To-One traffic pattern is used for the generation of the low-critical flow set. For this experiment, the size of the high-critical flow is 2 flits, while the size of a low-critical flow is 8 flits. The period and the release time of each low-critical flow are randomly generated.

Fig. 14 shows the rate of scheduled task of Rosace with different use rates of the network. Results show that DAS is more efficient on sharing resources than WNoC. For 10,71% of low-critical flows network use rate, 57% of tasks are scheduled using DAS, while using WNoC, we have just 3% of tasks. Using DAS, Rosace can be executed and respect its timing constraints with 7% of network low-critical flow use rate while WNoC provide just 5.7%.

6.3. Evaluation of system Mode Changes

The two previous experiments evaluate the flow latencies for different network workloads. Now, we increase the number of conflicts between flows, while maintaining the same network use rate. We assign the release times of high-critical flows to enforce those conflicts. In this experiment, we verify the ability of DAS to switch from normal mode to degraded mode.

We show in Fig. 15 the impact of the release times of a high critical flow on communication latencies. In the first case, the 2 flows share the same link in different time windows without having any interference. There is no preemption and no additional latency is expected for the two flows. In the second case, the high-critical flow arrives slightly before the release time of the low-critical flow. In this case, the two flows share the same link at the same time. The low-critical flow is delayed by the high-critical one and there is no preemption. At the third case, the high-critical flow arrives after the

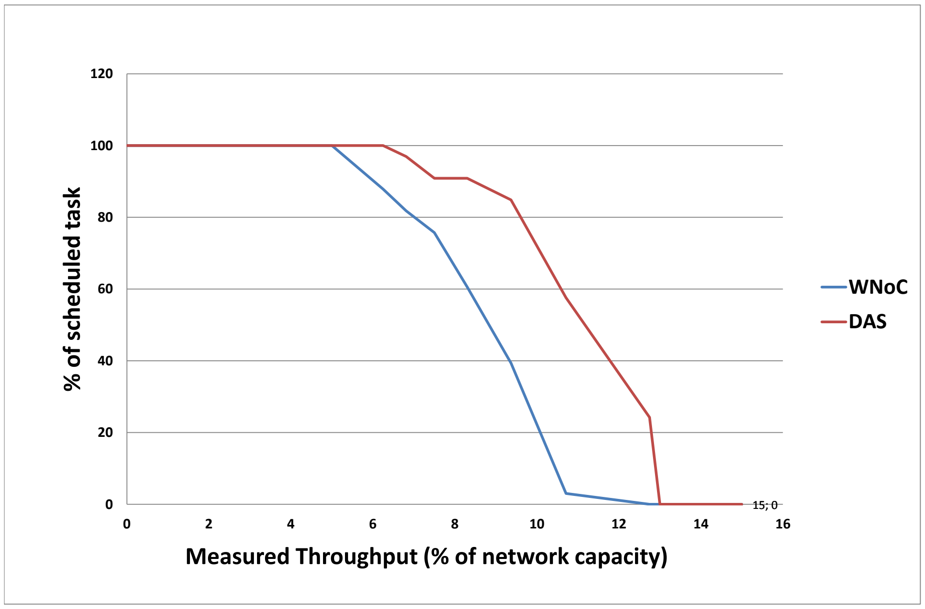


Fig. 14. Impact of communication time on application schedulability

release time of the low-critical flow. In this case, the high-critical flow preempts the low-critical flow, reducing WCCT for high-critical flows.

For this evaluation, we use a flow set where high-criticals and low-critical flows share the same physical link and use different virtual channels and different release times. The initial flow set is schedulable. In other word, each flow meets its deadline without having interference with other flows. It means that the system is initially in normal mode.

Then, to increase interferences, we change the release time of some flows and we compute the latency of each high-critical flow and the interference rate of the network. Here, simulations are performed with SHOC. We present the simulation results for 2 NoCs: a NoC with DAS and another one with a virtual channel router.

Fig. 16 and Fig. 17 show the latency of high-critical flows for 5% and 10% of network interference respectively. We group the flows following the length of their path, i.e., the number of hop. Fig. 16 and Fig. 17 present results of flows with 3, 4 and 5 hops.

For 5% of network interferences, DAS reduces more than 70% of the latency of high-criticality flows with 4 hops comparing with a VC router. For 10% of network interferences, DAS is able to reduce the communication delay for high-critical flows about 66%.

These results show that DAS is able to change from normal mode to degraded mode and to minimize the latency for high-critical flows when interferences occur within the NoC.

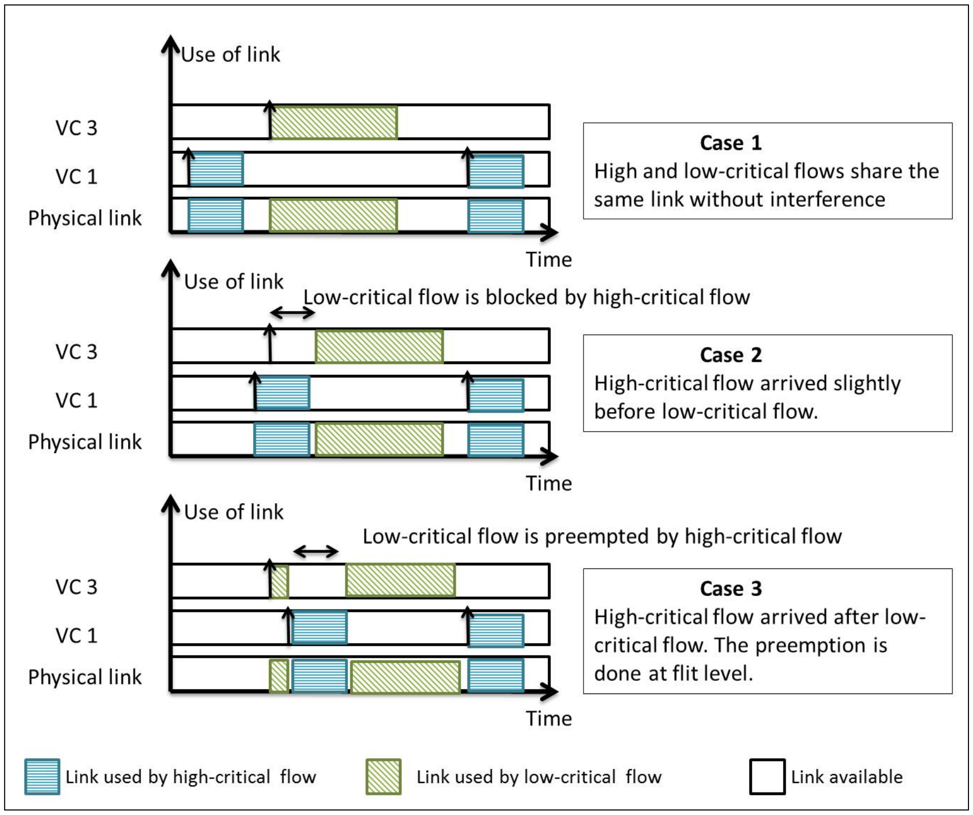

Use of link
VC 3
VC 1
Physical link
Time
Case 1
High and low-critical flows share the same link without interference
Low-critical flow is blocked by high-critical flow
Case 2
High-critical flow arrived slightly before low-critical flow.
Low-critical flow is preempted by high-critical flow
Case 3
High-critical flow arrived after low-critical flow. The preemption is done at flit level.
Link used by high-critical flow
Link used by low-critical flow
Link available

Fig. 15. Interferences depending on release times

6.4. Cost evaluation

We now present the results of the hardware synthesis of the DAS architecture that we made to evaluate its cost.

To evaluate the cost of DAS, three routers have been implemented. The first version is a NoC router without virtual channel. The second one is DAS. The last one is using VC routers. All routers have the same parameters, i.e., five bi-directional ports and 32-bits data width. They have the same cumulative buffer size (16-flit buffer size per port).

We have used Verilog-HDL for the circuit modeling. All routers have been synthesized with synopsys D using a *28 nm* ST SOI technology. Tools included in this technology generate reports describing the area of implementation.

The total cell area of all the 3 versions is presented in the Table V. The total area overhead of DAS is about 17% when compared with the NoC router without virtual channel and about 2,5% when compared with virtual channel router.

6.5. Formal verification of DAS

This last set of evaluations use a higher abstraction level. Simulation is the most used technique for design validation, however, when the design includes numerous parameters such as DAS, it is very hard to provide an exhaustive analysis. DAS properties are also expensive to validate on a real router implementation. In order to overcome these limitations, we use formal modeling and verification.

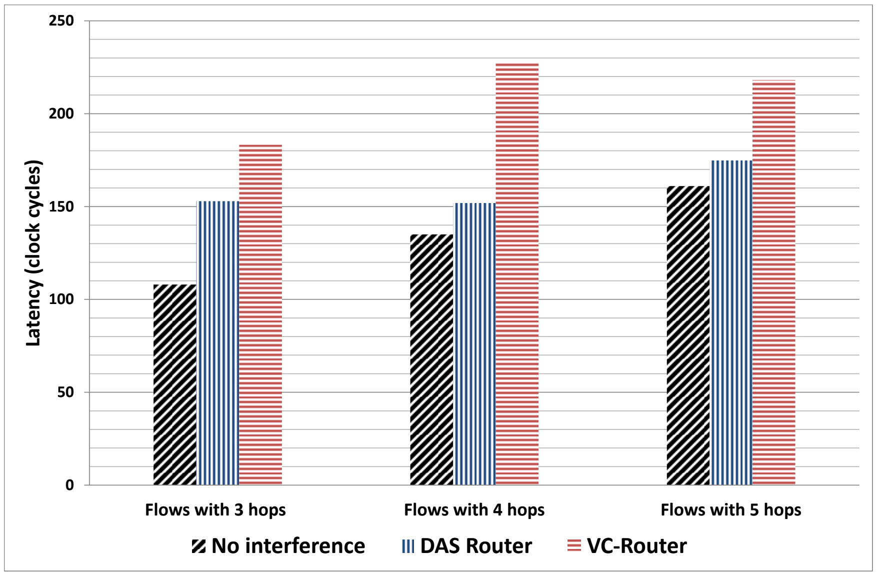


Fig. 16. 5% of network interference

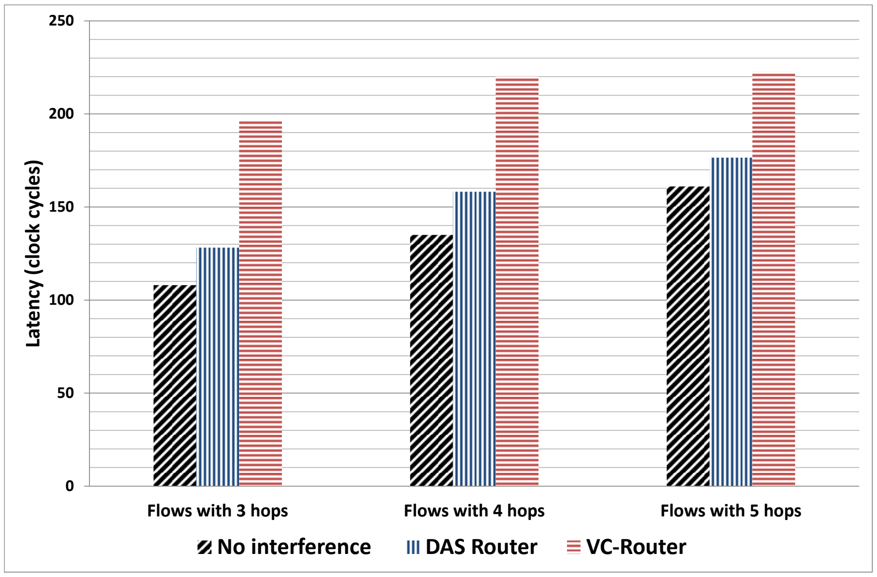


Fig. 17. 10% of network interference

Table V. Synthesis results of routers with Synopsys DC /28 nm ST SOI Technology

	NoC Router without Virtual Channel	**Virtual Channel Router**	**DAS**
Number of ports	5	5	5
Number of VCs	1	3	3
Data width	32-bit	32-bit	32-bit
Buffer's size per port	16-flit	16-flit	16-flit
Total cell area	16046.31	18369.47	18831.32

In this subsection, we discuss about the formal verification of DAS properties. First, details of DAS formal specification are given in IF language. Then, we explain the validation approach of this specification based on DAS properties and using the IF toolset. The validation approach is composed of interactive simulation and behavioral properties verification. Interactive simulations with several scenarios using IFx tool allow us to validate the proposed model. Then, the verification of behavioral properties of DAS presented in Section 5 is an exhaustive verification performed by IF observers.

6.5.1. Formal DAS Modeling in IF Language. DAS provides numerous functions. First, it accepts incoming messages. Second, it assigns for each incoming message a virtual channel depending on its criticality level. Third, the input arbitration unit chooses one message to be forwarded. Then, the switch unit calculates the destination port of the message. Finally, the corresponding output arbitration unit selects one message to be forwarded.

The overall architecture of the DAS Specification. The overall architecture of the router IF specification is presented in Fig 18. Here, DAS is modeled by an IF system. An IF system is composed of active process instances running in parallel and interacting asynchronously through shared variables and signals [Bozga et al. 2004]. Next routers and local processing elements are considered as environment entities of DAS model.

As shown in Fig. 18, the model system is composed of many entities: a main process, multiple instances of a child process (created by the main process after receiving an input message from the environment), one instance of switch, five instances of the first stage input arbiter (i.e., `input-arbiter-A`), the second stage input arbiter (i.e., `input-arbiter-B`), the first stage output arbiter (i.e., `output-arbiter-A`), and the second stage output arbiter (i.e., `output-arbiter-B`).

The main task of the main process is to create child process instances after receiving messages from the environment. The main task of the DAS child process is to compute the possible states of one message forwarded by DAS. The switch process selects the destination output port of the message. Input arbiters (i.e., `Input-arbiter-A` and `input-arbiter-B`) manage conflicts between flows which share the same input port. Similarly, output arbiters (i.e., `output-arbiter-A` and `output-arbiter-B`) manage conflicts between messages on the output port.

Figure 19 presents, as example, the input arbiter B (i.e., `input-arbiter-B`) automaton. [Dridi et al. 2017] provides more details about each entities.

Based on the overall architecture and the state machines of DAS entities, we expressed a formal specification of DAS in IF language. Below, we present some IF specification details, especially, data types, global variables, `signals` (messages) and `signalroutes` (communication buffers).

Data Types and Global Variables. In order to configure the DAS router specification, we define global constant parameters according to the number of child processes, arbiters and switches:

```
system DAS;
```

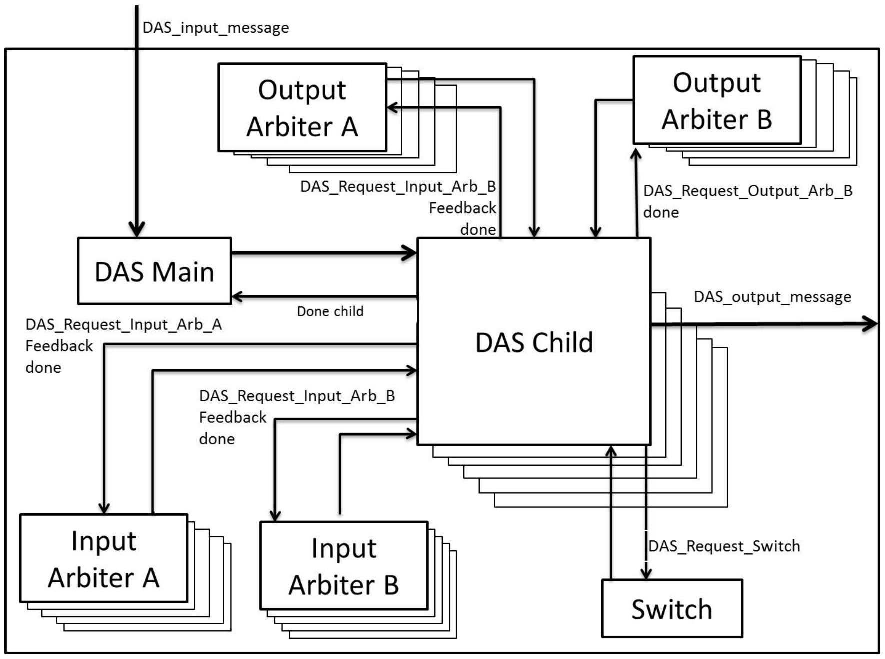


Fig. 18. Overall Architecture of DAS

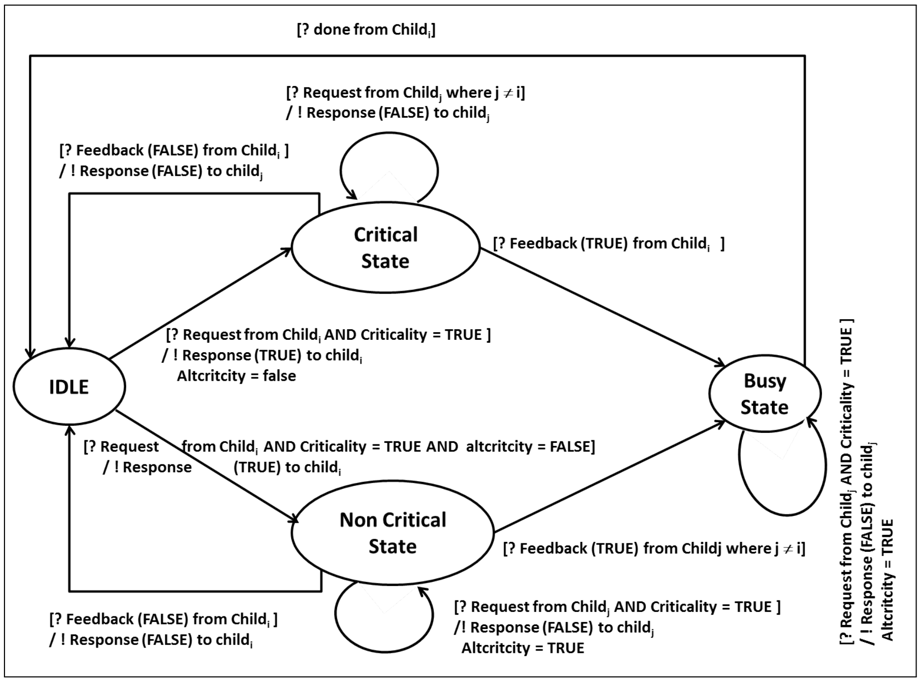

[? done from Child$_i$]
[? Request from Child$_j$ where j ≠ i] / ! Response (FALSE) to child$_j$
[? Feedback (FALSE) from Child$_i$] / ! Response (FALSE) to child$_j$
Critical State
[? Feedback (TRUE) from Child$_i$]
[? Request from Child$_i$ AND Criticality = TRUE] / ! Response (TRUE) to child$_i$ Altcritcity = false
IDLE
Busy State
[? Request from Child$_i$ AND Criticality = TRUE AND altcritcity = FALSE] / ! Response (TRUE) to child$_i$
Non Critical State
[? Feedback (TRUE) from Childj where j ≠ i]
[? Feedback (FALSE) from Child$_i$] / ! Response (FALSE) to child$_i$
[? Request from Child$_j$ AND Criticality = TRUE] /! Response (FALSE) to child$_j$ Altcritcity = TRUE
[? Request from Child$_j$ AND Criticality = TRUE] / ! Response (FALSE) to child$_j$ Altcritcity = TRUE

Fig. 19. The Input-Arbiter-B State Machine

```
 const M = 1; /* We have one DAS system */
 const N = 2; /* The number of child processes */
 const InA = 1; /* The number of Input Arbiter A processes */
 const InB = 1; /* The number of Input Arbiter B processes */
 const OutA = 1; /* The number of Output Arbiter A processes */
 const OutB = 1; /* The number of Output Arbiter B processes */
 const SW = 1; /* The number of Switch process */
  ...
endsystem;
```

These global parameters are used also to reduce the state space exploration during the DAS model verification. In addition, we define and use a global data structure (the table `childInfoTable`), to manage N active child process instances (i.e., virtual channels of the DAS router) and their information (e.g., instance `pid`, child table index, incoming message, the message criticality):

```
type DASMessageParameterType = record
 criticality boolean;
 InputAribtreChannelId InArbiterAIdType;
 OutputAribtreChannelId OutArbiterAIdType;
endrecord;

type ChildInfoMemberType = record
 childPid_exist boolean;
 childPid  pid;
 childIndex  IndexType;
 childMessage DASMessageParameterType;
 stocked   boolean;
 transferFormat TransferFormatType;
 SwitchChannelId SwitchIdType;
 InArbiterA_RequestStatus boolean;
 InArbiterB_RequestStatus boolean;
  ...
endrecord;

type ChildInfoTableType = array [N] of ChildInfoMemberType;
```

Signals and Signalroutes. We use signal types to model exchange messages. 15 signal types (e.g., `DAS_request_Input_Arb_B`, `done`, `feedback`) are defined to be used by the child process instances to communicate respectively with the environment, the different arbiters and the switch. For example, the incoming DAS router message `DAS_request_Input_Arb_B` is sent by the child process instance to the input arbiter A process instance.

13 `signalroutes` are used by the DAS specification as communication buffers between the environment and the different processes themselves. In the example below, the child process instance handles the signal `DAS_request_Input_Arb_B` to communicate with the input arbiter B via the `signalroute DASChild__to_Input_Arbiter_Stage_B`:

```
type IndexType = range 0 .. MaxOfChild_Index;

signal DAS_request_Input_Arb_B(IndexType);
signal feedback(boolean, IndexType);
signal done(boolean);

signalroute DASChild__to_Input_Arbiter_Stage_B(InB)
 from DASChild to Input_Arbiter_Stage_B
 with DAS_request_Input_Arb_B, feedback, done;
```

In the sequel, we present the validation of DAS formal specification. First, we use the IFx tool for interactive simulation with several scenarios presented in Table VI and for exhaustive formal validation of behavioral properties described by IF observers.

6.5.2. Validation with Simulations. In order to validate the main functions of DAS, we have simulated several scenarios. The different scenarios that were performed by interactive simulation using the IFx tool are shown in Table VI. It presents also the scenarios simulation results. The number of iterations for all scenarios is greater than 1 in order to validate that all the components (i.e., IF processes) return to the `idle` state.

The two first scenarios (categories 1 and 2) check the simple path of messages without any conflict in the network (i.e., the IF system and its environment). The other categories investigate scenarios with interferences between messages (arrived messages `DAS_input_message` in the main process of DAS specification). Categories 3, 4 and 5 verify the behavior of DAS against interferences between messages with the same criticality. Categories 6 and 7 verify the behavior of DAS router against interferences between messages with different criticality levels.

Finally, we note that all the simulated behaviors are consistent with the expected behaviors.

Table VI. Validation of DAS model by simulations: scenarios and results

Scenario Category	**Number of Child**	**Criticality (High/Low)**	**Input and output Channel Id**	**Number of Iterations**	**Validation Results**
Simple routing with high-critical	1	High	1, 2, 3 and 4	> 5	**Yes**
Simple routing with low-critical	1	Low	1, 2, 3 and 4	> 5	**Yes**
Arbitration between high-critical communication in input port	> 3	High	Same inputs Different outputs	> 5	**Yes**
Arbitration between high-critical communication in output port	> 3	High	Different inputs Same outputs	> 5	**Yes**
Arbitration between low-critical communication in output port	> 3	Low	Different inputs Same outputs	> 5	**Yes**
Flit-level Preemption in input port	3	1 High 2 Low	Same inputs Different outputs	> 5	**Yes**
Flit-level Preemption in output port	3	1 High 2 Low	Different inputs Same outputs	> 5	**Yes**

6.5.3. Exhaustive validation with IF Observers. After interactive simulation, we use the IFx tool and IF observers (provided by IF language) to describe and exhaustively verify DAS properties. We now exhibit proofs of the properties. More details about IF observers are given in [Dridi et al. 2017].

The validation results of the four properties given in the Section 5 are summarized in Table VII. It shows the number of states, the number of transitions, and the time taken by the IFx tool for an exhaustive exploration. We can notice that all explorations are terminated normally without moving to the `error` state (e.g., line 54 in the Figure 20) of IF observers and without any exploration cutting.

We use the IFx tool and IF observers formalism to describe and verify DAS properties. The observer illustrated in Figure 20 checks property 2.

This cut observer monitors the reception of the child process request by the input arbiter B in the `idle` state (lines 4,8). The observer keeps monitoring the sending of the response to the child process with boolean parameter (line 9), if the two internal variables `altCriticality` and criticality) are `false`, When this parameter is equal to false then the observer moves to an error state and cuts state exploration (lines 18-22, 24), else it moves to `idle` state (lines 15–17).

From those experiments, we can conclude that DAS model (described in IF language) satisfies the four behavioral properties of the system mode change.

Fig. 20. The IF Cut Observer of the Property 2

```
cut observer obs_prop_2;

 var index IndexType;
 var response boolean;
 var Output_Arbiter_Stage_B pid;

 state idle #start ;
 match input DAS_request_Output_Arb_B(index) in Output_Arbiter_Stage_B;
 nextstate input_DAS_request_Output_Arb_B_matched;
 endstate;

 state input_DAS_request_Output_Arb_B_matched;
 provided ({Output_Arbiter_Stage_B}0) instate availableState;
 nextstate check_criticity;
 endstate;

 state check_criticity;
 provided ({Output_Arbiter_Stage_B}0).criticity = false and ({Output_Arbiter_Stage_B}0).altCriticity=true;
 nextstate check_match_output_DAS_response_output_Arb_B_false;
 provided ({Output_Arbiter_Stage_B}0).criticity = false and ({Output_Arbiter_Stage_B}0).altCriticity=false;
 nextstate check_match_output_DAS_response_output_Arb_B_true;
 endstate;

 state check_match_output_DAS_response_output_Arb_B_false;
 match output DAS_response_Output_Arb_B(response);
 nextstate decision_false;
 endstate;

 state check_match_output_DAS_response_output_Arb_B_true;
 match output DAS_response_Output_Arb_B(response);
 nextstate decision_true;
 endstate;

 state decision_false #unstable ;
 provided (response = false);
 informal "−−Validation␣Success!";
 nextstate idle;
 provided (response = true);
 informal "−−Validation␣Fail!";
 cut;
 nextstate err;
 endstate;

 state decision_true #unstable ;
 provided (response = true);
 informal "−−Validation␣Success!";
 nextstate idle;
 provided (response = false);
 informal "−−Validation␣Fail!";
 cut;
 nextstate err;
 endstate;

 state err #error ;
 endstate;

endobserver;
```

Table VII. Exhaustive validation of DAS model by observers: State space and results. Those experiments were run on a Intel(R) Core(TM) i7-6700HQ CPU @ 2.60Ghz with 32GB RAM.

Properties	**Number of States**	**Number of Transitions**	**Time (hh:mm:ss)**	**Results**
Property 1	7 300 246	4 554 916	00:02:51	**Proved**
Property 2	1 823 025	566 238	00:00:43	**Proved**
Property 3	378 452	858 546	00:00:20	**Proved**
Property 4	356 684	811 734	00:00:18	**Proved**

6.6. Conclusion of DAS performance evaluation

Assuming small messages, DAS provides the best communication time for high-critical flow comparing with WNoC and VC router. DAS reduces by 80% the additional latency for a high-critical flow for 15% of network use rate and comparing to a VC router. DAS reduces by 64% the additional latency for a high-critical flow for 14% of network use rate and comparing to a WNoC. For high throughput in the network, DAS still provide the best communication time comparing with WNoC. The WCCT analysis is less pessimistic than the other solutions. In addition, while ensuring timing requirements of high-critical flows, DAS allows us to have the better throughput of low-critical flows in the network comparing with WNoC.

For big messages of high-critical flow, DAS losses its efficiency against WNoC. DAS does not provide the best communication time of low-critical flows comparing with other solutions such as WNoC and VC router. For 15% of network use rate and comparing to a VC router, DAS increases by 25% the additional latency for a low-critical flow. For 8% of network use rate and comparing to a WNoC, DAS increases by 21% the additional latency for a low-critical flow.

To conclude, DAS ensures the timing constraints of high-critical flows, minimizes the impact of sharing resources on low-critical flows and allows a non pessimistic WCCT analysis, which makes DAS suitable for the design of MCS.

6.7. Conclusion about multi-abstraction level evaluation

In this article, we evaluate DAS with several abstraction-level methods. We designed high-level models to perform formal verification with IF. We performed a hardware synthesis with Verilog HDL. We also made simulations using the SystemC SHOC simulator. Each abstraction-level evaluation has its own advantages and disadvantages.

Simulation based on a transaction-level modeling of routers in SystemC is characterized by fast time simulation comparing with other techniques. It is an efficient way of quickly evaluate early designs. However, most of the time, simulations cannot lead to exhaustive verification, and then, properties cannot be proved.

We also performed an hardware synthesis in order to verify the realism of DAS architecture from the circuit area point of view. In contrast to SystemC based simulations, hardware synthesis performs energy consumption and area cost evaluation, but it is an expensive method which cannot be applied as an early performance evaluation and design space exploration.

Finally, formal techniques can be helpful at higher levels of abstraction in a variety of ways. First, with formal techniques, we exhibit formal proofs of the four DAS properties related to the mode change protocol. It is also an opportunity to understand functional properties of the design. In our context, system mode change properties have been identified during formal specification with automata. Second, it helps to detect early errors of the design. Last, the formal specification allows us to detect areas for improvement and to implement the necessary actions. As an example with DAS, after formal specification with automata, we have understood that a virtual channel manager will improve efficiently the performances of the router.

To conclude, the three abstraction level verifications we performed for DAS was an opportunity to share parts of the models. Indeed, SystemC SHOC routers are implemented as automata that are close by IF language automata. It is then easier to implement System Noc router when their behaviors have been specified and proved with the IF language.

Those evaluations show that the 3 verification methods were complementary each others.

7. CONCLUSION

In mixed-criticality systems (MCS), applications with different levels of criticality share the same hardware platform. For that, NoCs for MCS must provide different operating modes in order to ensure the timing constraints for high-critical communications while limiting the bandwidth reservation for them, in order to improve the throughput for low-critical flows. In addition, we need an accurate communication time analysis in order to avoid the over reservation of resources for high-critical communications. To conclude, in order to deploy MCS over NoC, we need a router which (i) ensure the timing constraints for high-critical flows, (ii) minimize the impact of resource sharing on low-critical flows, (iii) and allow an accurate worst case communication time (WCCT) analysis.

In this article, we have proposed and evaluated a NoC router supporting MCS with an accurate WCCT analysis. The proposed router, called DAS (Double Arbiter and Switching router), jointly uses two switching modes: *Wormhole* and *Store And Forward*. On the assumption that small packets compose the high-critical communication traffic, the Store and Forward switching mode is used for high-critical flows. Wormhole policy remains for low-critical flows. DAS has two operating modes: a normal mode with both low and high-critical flows, and a degraded mode restricted to high-critical flows only. To move from on mode to another, DAS combines two levels of preemption: flit-level and packet-level.

To take advantage of many-core processors in real-time high-critical systems, a temporal analysis of their communication time is necessary. The NoC router architecture we propose helps for this analysis, while maintaining a high communication bandwidth. Multiple applications, with different timing requirements, could be carried out by such an execution platform, and then, the implementation of a real-time system should be improved from the integration and cost points of view. In order to decide about the schedulability of high-critical flows, a new analysis of WCCT for SAF communication with virtual channel is given. While, the WCCT analysis of Wormhole communication with shared policy is adapted for low-critical flows.

The second contribution of this paper is a multi-abstraction level evaluation of DAS. To evaluate DAS, we performed simulations with a SystemC simulator, we exhibit a formal proof by model-checking and we made an hardware synthesis. Both of the methods are complementary and allow us to the following results. First, we show a gain of 80% on latency for the high-critical flows, and a circuit size overhead limited to 2,5% to respect with a virtual channel router. Second, the evaluation of system mode change of DAS has shown a gain of 66% on high-critical communication delays for 10% of network interferences comparing with a virtual channel router. Finally, we have formally proved 4 properties regarding to the resources sharing between low and high-criticality flows in DAS.

Future works will address the task mapping according to the constraint of the number of virtual channels of DAS. We also plan to integrate a virtual channel manager into DAS in order to have a dynamic allocation of virtual channels. Finally, the WCCT model will be included into an end-to-end response time schedulability analysis method [Tindell and Clark 1994; Singhoff et al. 2004].

8. ACKNOWLEDGMENTS

This work and Cheddar[3] are supported by Brest Métropole, Ellidiss Technologies, CR de Bretagne, CD du Finistère and Campus France PESSOA programs number 27380SA and 37932TF.

[3]http://beru.univ-brest.fr/~singhoff/cheddar/

REFERENCES

H. Ahmadian and R. Obermaisser. 2015. Time-Triggered Extension Layer for On-Chip Network Interfaces in Mixed-Criticality Systems. In *2015 Euromicro Conference on Digital System Design*. 693–699. DOI:http://dx.doi.org/10.1109/DSD.2015.33

Mohammad Abdullah Al Faruque and Jorg Henkel. 2007. Transaction Specific Virtual Channel Allocation in QoS Supported On-chip Communication. In *Proceedings of the IEEE International Conf on Application-specific Systems, Architectures and Processors (ASAP)*. 48–53.

Mohammad Abdullah Al Faruque and Jorg Henkel. 2008. Minimizing Virtual Channel Buffer for Routers in On-chip Communication Architectures. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*. 1238–1243.

R. De Andrade, K. N. Hodel, J. F. Justo, A. M. Lagan, M. M. Santos, and Z. Gu. 2018. Analytical and Experimental Performance Evaluations of CAN-FD Bus. *IEEE Access* 6 (2018), 21287–21295. DOI:http://dx.doi.org/10.1109/ACCESS.2018.2826522

Enrico Bini and Giorgio C Buttazzo. 2005. Measuring the performance of schedulability tests. *Real-Time Systems* 30, 1-2 (2005), 129–154.

Marius Bozga, Susanne Graf, and Laurent Mounier. 2002. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV '02)*. Springer-Verlag, London, UK, UK, 343–348. http://dl.acm.org/citation.cfm?id=647771.734275

Marius Bozga, Susanne Graf, Iulian Ober, and Joseph Sifakis. 2004. The IF toolset. In *Formal Methods for the Design of Real-Time Systems. International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004. Revised Lectures (Lecture Notes in Comput. Sci. Vol. 3185)*. LNCS, Vol. 3185. Springer-Verlag, 237–267.

Alan Burns. 2014. System Mode Changes - General and Criticality-Based. In *Proceedings of the 2nd workshop on Mixed Criticality Systems (WMC)*. 3–8.

Alan Burns and Robert Davis. 2017. *Mixed criticality systems-a review, 9^{th} ed.* Technical Report. Department of Computer Science, University of York. http://www-users.cs.york.ac.uk/burns/review.pdf.

Alan Burns, James Harbin, and Leandro Soares Indrusiak. Dec 2014. A Wormhole NoC Protocol for Mixed Criticality Systems. In *Real-Time Systems Symposium (RTSS)*. IEEE.

Yean-Ru Chen, Wan-Ting Su, Pao-Ann Hsiung, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen. 2010. Formal modeling and verification for Network-on-chip. In *The 2010 International Conference on Green Circuits and Systems*. 299–304.

Mourad Dridi, Mounir Lallali, Stéphane Rubini, MJ Sepúlveda, Frank Singhoff, and Jean-Philippe Diguet. 2017. Modeling and Validation of a Mixed-Criticality NoC Router Using the IF Language. In *10th International Workshop on Network on Chip Architectures (NoCArc)*.

Mourad Dridi, Stéphane Rubini, Frank Singhoff, and Jean-Philippe Diguet. 2016. DTFM: a Flexible Model for Schedulability Analysis of Real-Time Applications on NoC-based Architectures. In *4th IEEE International Workshop on Real-Time Computing and Distributed systems in Emerging Applications (REACTION)*. 43–49.

R. Ernst and M. Di Natale. 2016. Mixed Criticality Systems A History of Misconceptions? *IEEE Design Test* 33, 5 (Oct 2016), 65–74. DOI:http://dx.doi.org/10.1109/MDAT.2016.2594790

Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. 2015. How Realistic is the Mixed-criticality Real-time System Model?. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS '15)*. ACM, New York, NY, USA, 139–148. DOI:http://dx.doi.org/10.1145/2834848.2834869

Soroosh Gholami and Hessam S. Sarjoughian. 2017. Modeling and Verification of Network-on-chip Using constrained-DEVS. In *Proceedings of the Symposium on Theory of Modeling & Simulation (TMS/DEVS '17)*. Society for Computer Simulation International, San Diego, CA, USA, Article 9, 12 pages.

Kees Goossens, John Dielissen, and Andrei Radulescu. 2005. Æthereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers* 22, 5 (Sept 2005), 414–421.

P. Graydon and I. Bate. 2013. Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. In *Proceedings of the Workshop on Mixed-Criticality Systems*. 19–24.

Leandro Soares Indrusiak, Alan Burns, and Borsilav Nikoli. 2016. *Analysis of buffering effects on hard real-time priority-preemptive wormhole networks*. Technical Report arXiv:1606.02942. https://arxiv.org/abs/1606.02942.

Leandro Soares Indrusiak, James Harbin, and Alan Burns. 2015. Average and Worst-Case Latency Improvements in Mixed-Criticality Wormhole Networks-on-Chip. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*. 47–56.

Kunj Jain, Sandeep K. Singh, Alak Majumder, and Abir J. Mondai. 2015. Problems encountered in various arbitration techniques used in NOC router: A survey. In *2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV)*. 62–67.

Krunal Jetly. 2013. *Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGAs*. Ph.D. Dissertation. University of Windsor.

Nikolay Kavaldjiev, Gerard J. M. Smit, and Pierre G. Ja nsen. 2004. A virtual channel router for on-chip networks. In *IEEE International SOC Conference, 2004. Proceedings.* 289–293. DOI:http://dx.doi.org/10.1109/SOCC.2004.1362438

Shaoteng Liu, Zhonghai Lu, and Axel Jantsch. 2015. Highway in TDM NoCs. In *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM.

Sheng Ma, Libo Huang, Mingche Lai, and Wei Shi. 2014. *NETWORKS-ON-CHIP From implementation to programming paradigms*. Morgan Kaufmann.

Ivan Miro-Panades, Fabien Clermidy, Pascal Vivet, and Alain Greiner. 2008. Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture. In *Proccedings of the second ACM/IEEE International Symposium on Networks-on-Chip (nocs2008)*. 139–148.

Fernando Moraes, Ney Calazans, Aline Mello, Leandro Muller, and Luciano Ost. 2004. HERMES: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal* 38, 1 (2004), 69 – 93. http://www.sciencedirect.com/science/article/pii/S0167926004000185

Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. 2014. The ROSACE case study: from Simulink specification to multi/many-core execution. In *Proceedings of the 20^{th} International Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 309–318.

Vinitha Arakkonam Palaniveloo and Arcot Sowmya. 2011. Application of Formal Methods for System-Level Verification of Network on Chip. In *2011 IEEE Computer Society Annual Symposium on VLSI*. 162–169.

M. Panic, C. Hernandez, E. Quinones, J. Abella, and F. J. Cazorla. 2016. Modeling High-Performance Wormhole NoCs for Critical Real-Time Embedded Systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–12. DOI:http://dx.doi.org/10.1109/RTAS.2016.7461342

M. Paulitsch, O. M. Duarte, H. Karray, K. Mueller, D. Muench, and J. Nowotsch. 2015. Mixed-Criticality Embedded Systems – A Balance Ensuring Partitioning and Performance. In *2015 Euromicro Conference on Digital System Design*. 453–461. DOI:http://dx.doi.org/10.1109/DSD.2015.100

D. Rahmati, S. Murali, L. Benini, F. Angiolini, G. De Micheli, and H. Sarbazi-Azad. 2013. Computing Accurate Performance Bounds for Best Effort Networks-on-Chip. *IEEE Trans. Comput.* 62, 3 (March 2013), 452–467. DOI:http://dx.doi.org/10.1109/TC.2011.240

M. S. Santos and R. d'Amore. 2018. Error detection method for the ARINC429 communication. In *2018 IEEE 19th Latin-American Test Symposium (LATS)*. 1–6. DOI:http://dx.doi.org/10.1109/LATW.2018.8349687

Martha Johanna Seplveda, Marius Strum, and Wang Jiang Chau. 2009. Performance Impact of QoSS (Quality-of-Security-Service) Inclusion for NoC-based Systems. In *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 12–14.

Zheng Shi. 2009. *Real-Time Communication Services for Networks on Chip*. Ph.D. Dissertation. University of York.

Zheng Shi and Alan Burns. 2008. Real Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. 161–170.

Zheng Shi and Alan Burns. 2009. Real-Time Communication Analysis with a Priority Share Policy in On-Chip Networks. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS)*. 3–12.

F. Singhoff, J. Legrand, L. Nana, and L. Marcé. 2004. Cheddar: a flexible Real-Time Scheduling Framework. *ACM SIGAda Ada Letters* 24, 4 (Dec 2004), 1–8.

Ken Tindell and John Clark. 1994. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming* 40, 2-3 (1994), 117–134.

Sebastian Tobuschat, Philip Axer, and Rolf Ernst. 2013. IDAMC: A NoC for mixed criticality systems. In *19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 149–156.

Sebastian Tobuschat and Rolf Ernst. 2017a. Efficient Latency Guarantees for Mixed-Criticality Networks-on-Chip. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 43–49.

S. Tobuschat and R. Ernst. 2017b. Real-time communication analysis for Networks-on-Chip with backpressure. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 590–595. DOI:http://dx.doi.org/10.23919/DATE.2017.7927055

Steve Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the* 28^{th} *International Real-Time Systems Symposium (RTSS)*. IEEE, 239–243.

Anam Zaman. 2015. *Formal verification of Network-on-Chip Architecture*. Ph.D. Dissertation. NUST, Islamabad, Pakistan.