Modeling and Validation of a Mixed-Criticality NoC Router Using the IF Language

Mourad Dridi[†],Mounir Lallali[†],Stéphane Rubini[†],Frank Singhoff[†],Jean-Philippe Diguet[‡]

[† ‡]Lab-STICC, CNRS, UMR6285
[†]Univ. Bretagne Occidentale, 29200 Brest, France
[‡]Univ. Bretagne Sud, 56100 Lorient, France
{mourad.dridi, mounir.lallali, stephane.rubini, frank.singhoff}@univ-brest.fr, jean-philippe.diguet@univ-ubs.fr

ABSTRACT

In Mixed-Criticality Systems (MCS), high-critical real-time and low-critical real-time applications share the same hardware platform. Today MCS must also be implementable on NoC-based architectures. Those applications exchange messages with different timing requirements through the same network. Sharing resources between flows in a NoC can lead to unpredictable latencies and subsequently complicate the implementation of MCS in many-core architectures. A solution is that NoC routers provide guarantees for high-critical communications with a minimum impact on performances for low-critical communications. We propose a new router called DAS, which exhibits such properties to support MCS applications. Moreover we introduce the first formal verification of the MCS properties of a NoC-router. We detail a formal specification of the DAS router, with the IF language, in order to verify its ability to support MCS applications. We also describe the validation approach of this specification based on those properties and using the IF toolset.

1. INTRODUCTION

In Mixed-Criticality Systems (MCS), applications with different levels of criticality share the same hardware [4].

High-critical real-time applications such as longitudinal flight controller have very stringent communication requirements. It is imperative to meet deadlines otherwise the whole system might fail. In contrary, low-critical real-time applications such as video decoder can tolerate some missed deadlines.

NoCs (Network-On-Chip) are widely used in many-core systems since they provide scalability, modularity, and communication parallelism. Many-core systems allow multiple applications to run at the same time in the same processor [9]. These applications exchange messages through the communication infrastructure. Sharing the communication infrastructure between flows of messages leads to additional delays which may damage the system.

NoCArc'17, October 15, 2017, Boston, MA, USA

ISBN 978-1-4503-5542-1/17/10…$15.00
DOI: https://doi.org/10.1145/3139540.3139543

In the context of MCS, NoC routers must provide latency guarantees for high-critical real-time communications with the minimum impact on performance on low-critical real-time communications.

We propose DAS Router (Double Arbiter and Switching Router), a new NoC router for MCS. It supports two criticality levels: high-critical flows and low-critical flows. It ensures timing constraints for high-critical flows while limiting the impact of sharing resources on low-critical flows [6].

To support MCS, DAS router must enforce the following properties:

- **P.1** High-critical flows always meet their timing constraints.
- **P.2** Low-critical flows are always able to exploit available resources in the network.
- **P.3** High-critical flows always preempt low-critical flows in flit-level.
- **P.4** High-critical flows always have a higher priority than low-critical flows.
- **P.5** Messages forwarded by DAS router can never be lost.

Those properties are expensive to validate on a real router implementation. Model-based verification methods aim to replace experimentations on a real prototype by validation on a formal model. In this paper, we show how to design a formal specification of DAS router in IF language and we describe the validation approach of this specification based on the DAS properties and using the IF toolset. The use of formal description of the DAS router in IF language results in an executable model that is used not only in interactive simulation but also in validation by observers (describing requirements) using the IF simulator.

The remainder of the paper is organized as follows. The section 2 introduces background elements about MCS, NoC, IF language and toolset. The section 3 presents the description of the DAS router and its modeling using IF language. Validation of the router specification is described in section 4. The section 5 deals with related works. Finally, section 6 concludes and outlines future works.

2. BACKGROUND

This section presents the necessary background on MCS, NoC, and the IF language and its toolset to understand DAS router formal specification.

2.1 Mixed-Criticality Systems

Mixed-Criticality Systems (MCS) are real-time systems characterized by two or more levels of criticality. In this work, we assume only two criticality levels: low-critical flow and

high-critical flow. In a MCS, high-critical real-time applications have very stringent communication requirements. It is mandatory that all packets generated by a high-critical flow are delivered before their deadline even under the worst case scenarios, while low-critical real-time flows can tolerate some delays in the communication service.

2.2 Network-On-Chip

A Network-On-Chip NoC is a network of nodes that can be processors, memories, peripherals or clusters of nodes. These units exchange messages through a communication infrastructure. The communication infrastructure is based on routers and links. The router is a the key component in a NoC. In this work, we focus on virtual channel routers. A virtual channel is an unidirectional logical connection between two nodes multiplexed with other virtual channels across the physical channel.

Routers forward messages using a switching mode which determines how a packet is allocated with buffers and channels and when it will receive service. In this work, we jointly use Wormhole and store and forward switching policies:

Store and forward policy (SAF). With SAF, each switch waits for the full packet to arrive before sending to the next router [9].

Wormhole policy. In a wormhole network, the packet is divided into a number of fixed size flits [9]. The packet is split into a header flit, one or several body flits and a tail flit. 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.

For virtual channel routers, many messages need the same input/output port at the same time using different virtual channels. Arbiters manage conflict between flows on I/O ports, There exist several arbitration mechanisms as round-robin or priority-based. Round-robin arbiter gives the lowest priority to the last served request in the next arbitration while priority-based arbiter chooses one packet from many requests based on their priority [8].

2.3 IF Language

A real-time system specification using IF language [2, 3] is composed of active process instances running in parallel and interacting asynchronously through shared signals (messages) and variables passing via signalroutes (buffers) or by direct addressing. An IF process instance describes sequential behaviors and can be created and destroyed dynamically during the system execution. It has a private FIFO buffer and local data (discrete variables and clocks). Each IF process is defined as a timed automaton extended with communication primitives, discrete data variables, and urgency attributes on transitions (deadlines). Each transition of this automaton has an enabling guard (on data variables and clocks) and a set of actions (i.e., signal inputs and outputs, process creation and destruction, and assignments).

2.4 IF Toolset, IFx Tool

The IF toolset [3] provides an environment for modeling and validation of real-time systems described in IF language. The core components of this toolset are the syntactic transformation component and the exploration platform. From the IF specification, the first component permits the construction of an abstract syntax tree which is a collection of C++ objects representing the syntactic elements present in the specification. The main features of the exploration platform are the simulation of the process execution (by using the abstract syntax trees) and the management of time and representation of the state space (by composing all the active processes). This exploration platform can be connected to different model-checking and test-case generation tools (e.g., CADP, TGV).

In the IF toolset, properties to be checked are specified (in an operational way) by observers. IF language provides observer constructs for every parts of a system (e.g., variables, states), elapsed time and observable system events including input and output events, forking of processes, etc. The IF observer is an extended timed automaton which is executed in parallel with the target system. This observer can be also used to cut selected executions paths (i.e., cut observer). It can also change the system state by modifying variables or sending signals. The communication between the observer process and the system is synchronous but this observer process has always the highest priority during exploration.

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

3. FORMAL MODELING OF DAS ROUTER

In this section, we present the modeling of the DAS router [6]. First, the new router is introduced. Then, the overall architecture is explained. Finally, details of its IF formal specification are given.

3.1 Presentation of the DAS Router

The first aim of the DAS router is to ensure that high-critical flows meet their deadline. The second aim is to limit the bandwidth reservation by high-critical flows in order to improve the network use rate for low-critical flows.

The architecture of the router is shown in Fig. 1. DAS router is composed of N+1 virtual channels (VC), input and output arbitration units, a routing logic, a virtual channel allocator, a switching allocator and a crossbar.

It combines two switching modes: on each port, the router applies a wormhole or a SAF switching depending on the criticality of the packets. N VCs are dedicated to high-critical flows that use SAF switching mode. The last VC is dedicated to the low-critical flows with a wormhole switching mode.

With SAF switching mode, each packet allocates only one link at a time. Then, considering small high-critical packets, the congestion can be controlled. Thus, we minimize the level of pessimism for the high-critical flows worst case communication time.

In the other hand, wormhole does not require large capacity buffers and reduces the communication latency for low-critical flows. High-critical flows preempt any low-critical flows on the last virtual channel at flit level.

As shown in Figure 1 (b), DAS router uses two stages of arbitration in order to provide a flit-level preemption for high-critical flows against low-critical flows, which allow us to enforce high-critical flows timing constraints.

[1]https://www.irit.fr/ifx/

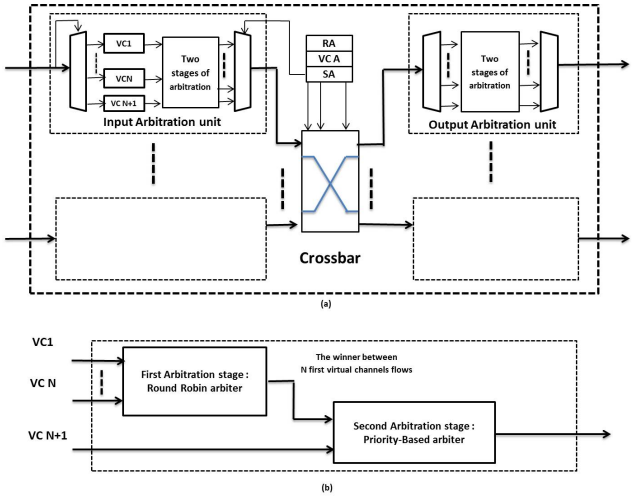

Figure 1: DAS Router: (a) Architecture, (b) Stages of arbitration

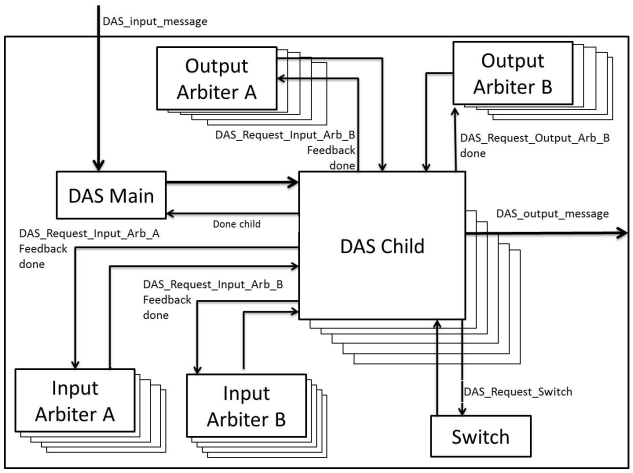

Figure 2: Overall Architecture of the DAS Router

3.2 Overall Architecture of the DAS Specification

The `DAS` router implements several functionalities in the network. First, it accepts new arrived messages. Second, it assigns a virtual channel to each message depending on its criticality level. Third, the input arbitration unit selects one message to be forwarded. After input arbitration, the switch computes the destination port of the message. Finally, the corresponding output arbitration unit chooses one message to be forwarded.

Figure 2 presents the overall architecture of the router `IF` specification. An `IF` system models the router. We consider next routers and local processing elements as environment entities of the system. As shown in Figure 2, the system is composed of a main process, one instance of switch, multiple instances of a child process (created by the main process after receiving an input message from the environment), and five instances of `input-arbiter-A`, `input-arbiter-B`, `output-arbiter-A`, and `output-arbiter-B`.

The `DAS` child process describes the possible states of one message forwarded by the `DAS` router. The main task of the switch process is to compute the destination output port of the message. `Input-arbiter-A` and `input-arbiter-B` manage conflicts between messages which share the same input port. Similarly, `output-arbiter-A` and `output-arbiter-B` manage conflicts between flows on the output port.

In the sequel, we describe the behavior of each of these entities.

3.2.1 The Main Process

If the main process receives the `DAS_input_message` messages from the environment, i.e. messages are sent from the environment. It creates a child process instance. For high-critical flows, a child models a virtual channel. However, for low-critical flows, a child models one flit using the last virtual channel. As a `DAS` router may have several flows, we have several instances of the child process.

3.2.2 The Child Process

We now describe the behavior of the child process as shown in Figure 3.

A child process has 10 states (`Q1`–`Q10`). It uses 5 internal boolean variables (e.g. `InArbiterA_Response`) related to the switch and arbiter responses for the child process requests.

The automaton starts at `Q1`. From state Q1, high-critical children, which use `SAF` switching mode, go to the state `Q3` while low-critical children go to the state `Q2`.

Once a message is stored in the router, a child requests `input-arbiter-A` and/or `input-arbiter-B` by passing over state `Q4` and `Q5` depending on its criticality. In state `Q6`, the child waits for the response of `input-Arbiter-B`, then, it advances to `Q7` and the switch computes the destination output port. Once the output port is known, the child requests the corresponding `output-Arbiter-A` and/or `output-Arbiter-B` by passing over state `Q7` and `Q8`. We note that in `Q7` and `Q8`, the children send `feedback` signals to input arbiters in order to inform them about the state of the message.

The message will be sent to the environment and the child is killed when the corresponding input and output arbiters are available.

3.2.3 The Input Arbitration Unit

Let see now the input arbitration units. At each cycle, for each `input-channel-id`, only one child can advance. However, many children with the same `input-channel-id` may ask to advance to a destination output port. The main task of an input arbitration unit is to choose one child among them for each `input-channel-id`.

The input arbitration units are composed of two stages of arbitration: `input-arbiter-A` and `input-arbiter-B`. `input-arbiter-A` is a fair arbitration between all high-critical children. `input-arbiter-B` is a fixed priority between high-critical children

Figure 4 presents the `input-arbiter-B` automaton. It starts in the `idle` state and then moves to the `critical` state when high-critical children request the arbiter and to the `non critical` state otherwise. It switches to the `busy` state when it receives a `feedback(true)` from the corresponding child. Once the message is forwarded, `input-arbiter-B` returns to the `idle` State. As shown in Figure 4, we note that high-critical children have a higher priority than low-critical children.

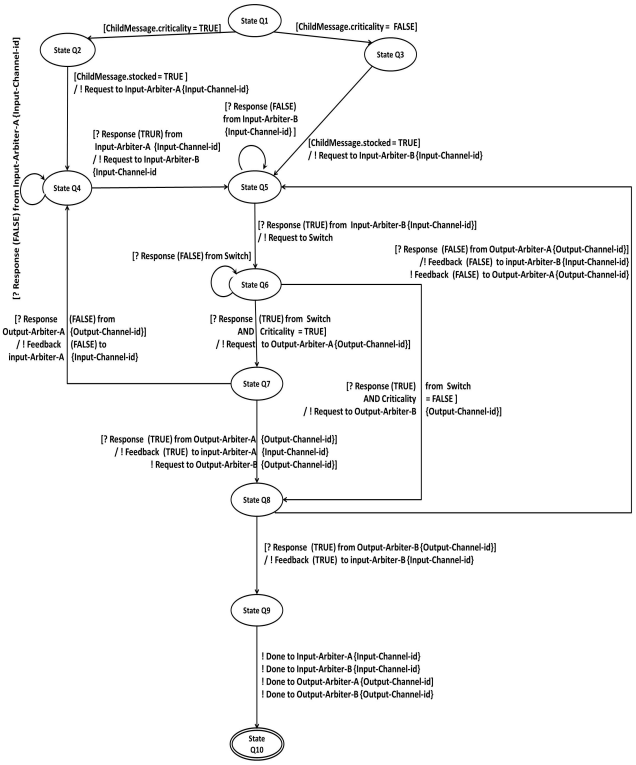

Figure 3: The Child Process State Machine

3.2.4 *The Output Arbitration Unit*

The main task of output arbitration is to choose one child for each `output-channel-id`. Similarly to input arbitration unit, an output arbitration unit is composed of two stages of arbitration: `output-arbiter-A` and `output-arbiter-B`. `output-arbiter-A` is a fair arbitration between all the high-critical children coming from different input ports and requesting the same output port. `output-arbiter-B` is a fixed priority arbiter.

3.3 Formal DAS Modeling in IF Language

Based on the overall architecture and the state machines described in the previous section, a formal specification in `IF` language of the `DAS` router is given by using the `IF` Toolset. Below, some `IF` specification details are presented, especially, the data types, global variables, `signals` and `signalroutes`.

3.3.1 *Data Types and Global Variables*

In order to configure the `DAS` router specification, global constant parameters are defined as the number of child processes and arbiters:

```
system DAS;
 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; const OutA=1; const OutB=1;
 const SW=1;   /* The number of Switch process */
 ...
endsystem ;
```

These parameters are used to reduce the state space exploration during the model validation (see Section 4).

In addition, the table `childInfoTable` is defined and used to manage the number of `N` active child processes (i.e., virtual channels of the `DAS` router) and their information (e.g., `pid`, child table index, 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  ;
```

3.3.2 *Signals and Signalroutes*

The last elements of our `IF` specification are the signals it handles. We use 14 `signal` types to model exchange messages. 13 `signal` types (e.g., `DAS_output_message`) are used by the child processes to communicate respectively with the environment, the different arbiters and the switch. The arrived `DAS` router message `DAS_input_message` is sent by the environment to the main process. 13 `signalroutes` are used by the `DAS` specification as communication buffers between the environment and the different processes themselves.

In the example below, the chid process handles `DAS_request_Input_Arb_A` signal to communicate with the `input arbiter A` via the `signalroute DASChild__to_Input_Arbiter_Stage_A`:

```
signalroute    DASChild__to_Input_Arbiter_Stage_A   (InA)
 from  DASChild  to   Input_Arbiter_Stage_A
 with   DAS_request_Input_Arb_A  ,  feedback ;
```

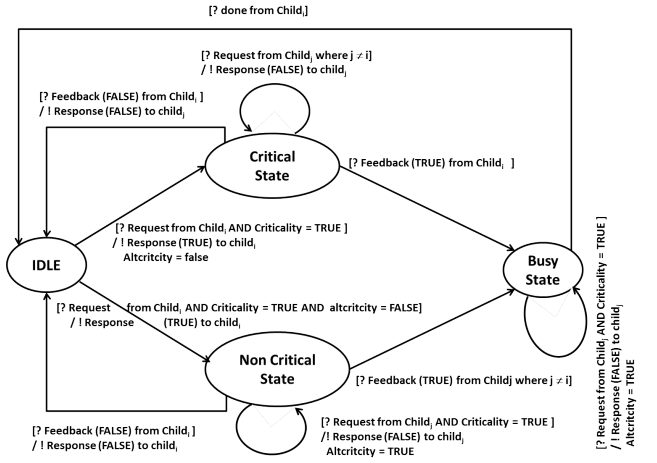

Figure 4: The Input-Arbiter-B State Machine

4. VALIDATION OF THE DAS ROUTER SPECIFICATION

In this section we discuss about the validation of the `DAS` router formal specification. We use the `IFx` tool for interactive simulation with several scenarios and for formal validation of behavioral properties described by `IF` observers.

4.1 Validation with Simulations

We have simulated several scenarios in order to validate the main functionalities of the `DAS` router.

Table 1 shows the different scenarios that we realized by interactive simulation using the `IFx` tool. 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.

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 the DAS router specification). Categories 3, 4 and 5 verify the behavior of DAS router 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 level.

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

4.2 Validation with IF Observers

IF language provides observer constructs. After interactive simulation, we use IF observers to specify (operationally) and verify DAS router properties on the formal DAS specification.

4.2.1 Requirements and properties

As in [1], we have identified a set of functional requirements classified into three classes: general requirements (not specific to the DAS router), global system requirements (concerning the global behavior of the router), and local component requirements (specific to some parts of the router, e.g. process or state).

Below, we give five examples of DAS router properties. The property 5 may be considered as a general requirement while properties 1–4 are considered as global requirements:

P.1 High-critical flows always meet their timing constraints.
P.2 Low-critical flows are always able to exploit available resources in the network.
P.3 High-critical flows always preempt low-critical flows in flit-level.
P.4 High-critical flows always have a higher priority than low-critical flows.
P.5 Messages forwarded by DAS router can never be lost.

4.2.2 Properties Validation With IF Observers

We use the IFx tool and IF observers formalism to describe and verify DAS properties. The observer illustrated in Figure 5 checks property 3. This cut observer starts monitoring the reception of the child process request by the input arbiter B in the idle state (lines 4,8). If the two internal variables (altCriticality and criticality) are false, then the observer keeps monitoring the sending of the response to the child process with boolean parameter (line 9). When this parameter is equal to true then the observer moves to idle state (lines 15–17), else it moves to an error state and cuts state exploration (lines 18-22, 24). By using the IFx tool to verify the property 3, we follow the next steps: (i) describing the property by editing the cut observer process das-prop3.oif in IF language, (ii) generating the IF simulator by compiling the IF model das.if to the executable file das.x using the command if2gen and the input associated observer, (iii) executing the generated simulator with partial order reduction, and depth-first-search (-dfs) options.

Inspired by [7], to avoid any state explosion and to reduce the problem size during validation, we limit the number of active child process to 3 and the number of each arbiter (input A/B, output A/B) to 2. The validation results of properties 3 and 4 are summarized in Table 2 that shows the number of states, the number of transitions, and the time taken by the IF simulator for an exhaustive exploration. We can notice that all explorations are terminated normally without moving to the error state (of the observer) and without any exploration cutting.

From those experiments, we can conclude that our DAS router model (described in IF language) satisfies the properties 3 and 4 for the proposed configuration (i.e., 3 active child processes, one switch process and 4 arbiters).

Those experiments were run on a Intel(R) Core(TM) i7-6700HQ CPU @ 2.60Ghz with 32GB RAM.

Figure 5: The IF Cut Observer of the Property 3

```
1   cut observer obs_prop_3 ;
2   var response boolean ; var index IndexType ; var Input_Arbiter_Stage_B pid ;
3   state idle #start ;
4       match input DAS_request_Output_Arb_B ( index ) in Input_Arbiter_Stage_B ;
5       nextstate input_DAS_request_Input_Arb_B_matched ;
6   endstate ;
7   state input_DAS_request_Input_Arb_B_matched ;
8       provided (({ Input_Arbiter_Stage_B }0) instate idle and
              ({ Input_Arbiter_Stage_B }0). altCriticality = false and
              ({ Input_Arbiter_Stage_B }0). criticality = false);
9           match output DAS_response_Input_Arb_B ( response );
10          nextstate decision_1 ;
11      provided ...
12          nextstate decision_2 ;
13  endstate ;
14  state decision_2 #unstable ;
15      provided ( response = true);
16          informal "--Validation_Success!";
17          nextstate idle ;
18      provided ( response = false);
19          informal "--Validation_Fail!";
20          cut ;
21          nextstate err ;
22  endstate ;
23  state decision_2 #unstable ; ... endstate ;
24  state err #error ; endstate ;
25  endobserver ;
```

5. RELATED WORKS

The most used technique for NoC verification is simulation. However, simulations cannot lead to proofs of the properties. In order to overcome this limitation, several works propose functional or/and performances analysis of NoC using formal methods.

[11] designs a formal model of the existing HERMES NoC router architecture [10] and its communication scheme using Heterogeneous Protocol Automota (HPA). It checks the functional properties of the communication architecture and uses a tool for modeling and verifying called Simple Promela Interpreter (SPIN). In [5] authors verify four crucial properties of an NoC router, namely, mutual exclusion, starvation freedom, deadlock freedom, and conditions for traffic congestions. This work also validates the bidirectional channel Network-on-Chip (BiNoC). It uses a formal verification model checking tool State Graph Manipulators (SGM) to perform such verifications.

[12] proposes a functional and performance analysis of both circuit switched and packet switched NoCs using formals methods. For the packet-switched NoC, they choose HERMES, while, for the circuit switched the Programmable NoC (PNoC) [12] is chosen. They apply the SPIN model checker to verify properties such as mutual exclusion, starvation freedom, deadlock and livelock.

All previous works verify existing NoCs with formal methods, but none of them support MCS applications. Thus, NoC properties such as flit-level preemption between flows with

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 (1, 2, 3 and 4) Different outputs (1, 2, 3 and 4)	> 5	**Yes**
Arbitration between high-critical communication in output port	> 3	High	Different inputs (1, 2, 3 and 4) Same outputs (1, 2, 3 and 4)	> 5	**Yes**
Arbitration between low-critical communication in output port	> 3	Low	Different inputs (1, 2, 3 and 4) Same outputs (1, 2, 3 and 4)	> 5	**Yes**
Flit-level Preemption in input port	3	1 High 2 Low	Same inputs (1, 2, 3 and 4) Different outputs (1, 2, 3 and 4)	> 5	**Yes**
Flit-level Preemption in output port	3	1 High 2 Low	Different inputs (1, 2, 3 and 4) Same outputs (1, 2, 3 and 4)	> 5	**Yes**

Table 1: Validation of DAS Router Model by Simulations: Scenarios and Results

Properties	Number of States	Number of Transitions	Time (hh:mm:ss)	Results
P.3	378452	858546	00:00:20	**Validate**
P.4	356684	811734	00:00:18	**Validate**

Table 2: Validation of DAS Router Model by Observers: State Space and Results

different levels of criticality were never investigated. However, the verification of these properties is absolutely necessary for the adoption of NoC-based architecture in MCS. In this paper, we address this shortfall. We focus on the formal validation of a NoC router supporting MCS and then, we investigate new properties ranging from flit-level preemption, deadlines met by flows or priority order on resource accesses.

6. CONCLUSION

In this paper, we show how to verify crucial properties of a hardware design with formal methods. We focus on the modeling and on the formal validation of a NoC router, called DAS router, which supports MCS applications. First, we give a formal specification in IF Language of the DAS router. Second, we present the validation of this specification using the IFx tool (an extended version of the IF toolset). Simulations of several scenarios are given by using the IF simulator. Formal validation of a NoC router supporting MCS requires us to investigate properties ranging from flit-level preemption, deadlines met by flows or priority order on resource accesses. Then, we identified five properties that the DAS router has to met and we verified some of them using IF observers.

For future works, we have to validate other identified properties of the DAS router.

7. ACKNOWLEDGMENTS

This work and Cheddar[2] (a GPL real-time scheduling analyzer) are supported by Brest Métropole, Ellidiss Technologies, CR de Bretagne, CG du Finistère and Campus France PESSOA programs number 27380SA and 37932TF.

8. REFERENCES

[1] M. Bozga, I. Graf, I. Ober, and J. Sifakis. The IF toolset. In *4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, SFM-04:RT, Bologna, Sept. 2004*, volume 3185 of *LNCS Tutorials*. Springer Verlag, 2004.

[2] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In *Proceedings of CAV'02*, pages 343–348, London, UK, 2002. Springer-Verlag.

[3] M. Bozga, S. Graf, I. Ober, and J. Sifakis. The IF toolset. In *SFM-04*, volume 3185 of *LNCS*, pages 237–267. Springer-Verlag, June 2004.

[4] A. Burns and R. Davis. Mixed criticality systems-a review, 9^{th} ed. Technical report, Department of Computer Science, University of York, Jan 2017. http://www-users.cs.york.ac.uk/burns/review.pdf.

[5] Y. R. Chen, W. T. Su, P. A. Hsiung, Y. C. Lan, Y. H. Hu, and S. J. Chen. Formal modeling and verification for network-on-chip. In *The 2010 International Conference on Green Circuits and Systems*, pages 299–304, June 2010.

[6] M. Dridi, S. Rubini, M. Lallali, M. Johanna, F. Singhoff, and J.-P. Diguet. Das: an efficient noc router for mixed-criticality real-time systems. In *2017 IEEE International Conference on Computer Design*, November 2017.

[7] I. Hwang, M. Lallali, A. R. Cavalli, and D. Verchère. Modeling, validation, and verification of pcep using the if language. In *FMOODS/FORTE*, pages 122–136, 2009.

[8] K. Jain, S. K. Singh, A. Majumder, and A. J. Mondai. Problems encountered in various arbitration techniques used in noc router: A survey. In *2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV)*, pages 62–67, Jan 2015.

[9] K. Jetly. *Experimental Comparison of Store-and-Forward and Wormhole NoC Routers for FPGA*. PhD thesis, University of Windsor, Nov 2013.

[10] F. Moraes, N. Calazans, A. Mello, L. Muller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, 38(1):69 – 93, 2004.

[11] V. A. Palaniveloo and A. Sowmya. Application of formal methods for system-level verification of network on chip. In *2011 IEEE Computer Society Annual Symposium on VLSI*, pages 162–169, July 2011.

[12] A. Zaman. *Formal verification of Network-on-Chip Architecture*. PhD thesis, NUST, Islamabad, Pakistan., August 2015.

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