Model-based Design and Formal Analysis of Arbitration Protocols on Multiple-Bus Architecture

Imene Ben-Hafaiedh[1] and Maroua Ben Slimane[2]

[1] Higher Institute of Computer Science (ISI), LIP2 Lab
[2] Tunisia Polytechnic School, LIP2 Lab

Abstract. Multiple-Bus architectures for multiprocessors systems are preferred by manufacturers for implementation as they are providing higher bandwidth and more reliability than single bus architectures. The key notion in such a multiple-bus system is the bus arbitration strategy which resolves request conflicts and allocates buses to requesting devices. Indeed, fast and fair bus arbitration gives higher throughput and less queuing time. In this paper we propose a high-level formal model of multiple-bus multiprocessor architecture seen as a component-based system. The proposed model offers a way to easily implement, analyze and compare different arbitration protocols using a rich notion of connectors between components. This model allows to study relevant properties such as fairness and deadlock freedom at a high level of abstraction. Moreover, the proposed model of the studied protocols is modeled in a distributed manner which makes the generation of their implementation more interesting and the study of their performance efficiency more relevant.

Key words: Formal Model, Multiple-bus architecture, Distributed Model, Arbitration Protocols

1 Introduction

In multiprocessors systems (MPSoC), heterogeneous components (processors, memories, logics, DSPs...) are integrated into a single chip. That is why the performance of these systems depends more on efficient communication among these devices. In general, this communication ensures access to shared resources. In the case of multiple bus architectures, such devices share a limited number of buses. The access to these shared buses is managed by a bus arbiter that resolves request conflicts and allocates buses to requesting devices. The efficiency of these systems depends considerably on the protocol used by the arbiter for bus allocation and how this protocol could be balanced and time efficient. Thus, the choice of the arbitration protocol plays an essential role in deciding the performance of bus based systems [1,2,3,4].
Multiple bus systems performance could be guaranteed by adopting effective bus arbitration protocols (policies) which could ensure several requirements such as

deadlock freedom, fairness, simplicity and time efficiency [5,1]. Analyzing properties of these different protocols at a high-level system description is useful for comparing their performance without having to go into very low-level implementation details. We claim that arbitration protocols could be naturally specified by expressive high-level features such as *Connectors* and *Glues* specifying rich interaction models between the different devices. Such features are provided by several component-based frameworks like those described in [6,7,8]. In [7], such rich interaction models allow the description of the different protocols based on high-level expressive notions like multi-party interactions and global priorities. The use of these notions leads to much more concise specifications which are more adequate to provide a comprehension of the global behavior of the different protocols and for the verification of their global properties. In [9], a similar approach has been proposed, however this approach was limited to a single bus architecture. In the present work, we propose a high-level analysis approach for systems with *multiple-bus* multiprocessors architecture which makes the arbitration protocols completely different with more interesting issues to take into account. Moreover in such multiple bus architectures, distributed configuration and implementation become more interesting.
Indeed, in multiple bus protocols there are two types of conflicts to manage. First, a conflict between processors to get a bus access. Second, a conflict between buses, which is not taken into account in [9]. In this paper, our first aim is to propose a way allowing to decide at a high-level about the appropriate arbitration protocol to implement. This choice is in general related to a set of requirements and properties (fairness, time efficiency, ...). Thus if such properties could be studied at a high-level, one can decide about the protocol to use without having to go into low level implementation details.
To reach this purpose, we propose an abstract formal model of multiple-bus architecture using a component based framework named BIP (Behavior-Interaction-Priority) [7,10]. This framework offers an expressive interaction model based on a rich notion of glues called *connectors* [11] [12] and an interesting notion of priority between interactions [13].

Based on these notions, we propose a model to describe, in an elegant manner the multiple bus architecture for three well-known arbitration protocols namely fixed, equal and rotating priority protocols.

The second contribution of this paper is the distributed aspect of the proposed model. In fact, our model is designed in a distributed manner, which means that a set of local arbiters, associated each to a device, control the access to the different buses. Such a model makes a distributed implementation easily generated. Thus a set of experimental results are then easily obtained to allow protocols efficiency studies.

The rest of this paper is organized as follows: In Section 2, we give an overview about the component-based (BIP) framework adopted to design the proposed high-level model. Then, in Section 3, we describe our proposed abstract and formal model for multiple bus architecture and how this model can be parameterized to define several well-known arbitration protocols. Section 4 presents some experimental results obtained from our automatically derived implementation. In Section 5, we compare our work to some existing researches. In Section 6, we sum up and discuss possible perspectives.

2 BIP Framework

In this section, we present a high-level modeling formalism for the description a multiple bus network architecture. We choose to specify our model using the BIP (Behavior-Interaction-Priority) component framework [7] [12] as it is a framework with formal semantics that relies on rich interaction models between components [11]. These interaction models are based on multi-party *interactions* for synchronizing *components* and in particular *priorities* for scheduling between interactions [13]. Moreover, this framework provides different tools for property verification and distributed code generation.

Definition 1 (Atomic component). *An atomic component K is a* Labeled Transition System *(LTS) defined by a tuple (Q, V, P, δ, q_0) where Q is a set of states,V is a finite set of variables, P is a set of communication ports, a transition relation $\delta \subseteq Q \times P \times Q$ and $q_o \in Q$ is the initial state.*
As usually, $q_1 \xrightarrow{a} q_2$ denotes $(q_1, a, q_2) \in \delta$

In practice, each variable may be associated to a port and modified through interactions involving this port. We also associate a *guard* and an *update function* to each transition. A guard is a predicate on variables that must be true to allow the execution of the transition. An update function is a local computation triggered by the transition that modifies the variables. Atomic components in BIP interact using *interactions.*

Definition 2 (Interaction). *Given a set of n atomic components $K_i = (Q_i, V_i, P_i, \delta_i, q_{o_i})$ for $i \in [1, n]$, in order to build their composition, we require their sets of ports to be pairwise disjoint, i.e. for any two $i \neq j$ from $[1, n]$, $P_i \cap P_j = \emptyset$. The composed system is defined on $P = \bigcup_{i=1}^{n} P_i$ and the set of its ports is defined by a set of* interactions. *An interaction a is defined by a non empty subset of P representing a* joint transition *of the set of transitions labeled by these ports. Note that each interaction defines itself a port of the composition which is useful for defining systems hierarchically (Definition 3).*

Composition of components allows to build a system as a set of components that interact by respecting constraints of an interaction model. In BIP interactions are structured by *connectors.* A *connector* is a macro notation for representing sets of related interactions in a compact manner. To specify the interactions of a connector, two types of synchronizations are defined:

– strong synchronization or *rendez-vous*, when the only interaction of a connector is the maximal one, i.e., it contains all the ports of the connector.

– weak synchronization or *broadcast*, when interactions are all those containing any port initiating the broadcast.

To characterize these two types of synchronizations, a connector may associate to the ports it connects two types:

– A *trigger* port of a connector is a complete port which can initiate an interaction without synchronizing with other ports of the connector. It is represented graphically by a triangle.
– A *synchron* port of a connector which is an incomplete port, hence needs synchronization with other ports, and is denoted by a circle.

Let γ be a connector connecting a set of ports $\{p_i\}_{i=1}^n$, then γ is defined as follows:

Definition 3 (Connector). *A connector γ is defined as a tuple $(p_\gamma[x], \mathcal{P}_\gamma, \delta_\gamma)$ where:*

– *$p_\gamma[x]$ is a port called the* exported *port of γ with x as associated variable,*
– *$\mathcal{P}_\gamma = \{p_i[x_i]\}_{i=1}^n$ is the set of connected ports called the* support set *of γ. These ports are typed by the information whether they are* trigger *or* synchron *port. x_i is a variables associated to p_i.*
– *$\delta_\gamma = (G, U, D)$ where,*
  - *G is a guard of γ, an arbitrary predicate $G(\{x_i\}_{i\in[1,n]})$,*
  - *$\mathcal{U}$ is an upward update function of γ of the form, $x := F^u(\{x_i\}_{i\in[1,n]})$,*
  - *$\mathcal{D}$ is a downward update function of γ of the form, $\cup_{p_i}\{x_i := F^d_{x_i}(x)\}$.*

The intuition behind the notion of exported port, as illustrated in Figure 1, is that a connector allows to relate a set of inner ports (of connected components) to a new port (the exported port) which allows to provide a notion of encapsulation by defining the interface of the obtained composite component. The composition

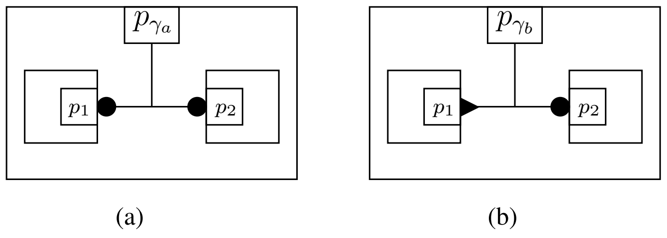

Fig. 1. Example of Connectors.

of BIP components with a connector γ defines an LTS (Definition 4).

Definition 4 (Composition of components). *The composition of n components K_i by a connector γ is denoted by $K = \gamma(K_1, \ldots, K_n)$ and it defines an LTS (Q, P, δ) such that $Q = \prod_{i=1}^n Q_i$ and δ is the least set of transitions satisfying the following rule:*

$$\frac{a = \{p_i\}_{i\in[1,n]},\ \forall i \in [1,n].\ q_i^1 \xrightarrow{p_i} q_i^2 \wedge\ \forall i \notin [1,n].\ q_i^1 = q_i^2}{(q_1^1, \ldots, q_n^1) \xrightarrow{a} (q_1^2, \ldots, q_n^2)}$$

The Definition 3 of connectors represents connectors which are essentially flat, i.e., having types (triggers and synchrons) associated to the individual ports (support set) only. However, connectors sometimes need to be structured, i.e., having types associated to groups of ports. This is necessary to represent some interactions, which otherwise cannot be represented by a flat connector. Structured connectors are created by the combined mechanism of exporting port from a connector and instantiating connectors, where a port of the connector is an exported port of another instantiated connector.

This is called domination [11]. That is, γ_i dominates γ_j means that the exported port of γ_j belongs to the support set of γ_i (see Figure 2).

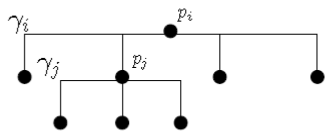

Fig. 2. Dominance between connectors.

The purpose of this work is to provide a formal high level model of multiple bus architecture. Using the different features offered by BIP, in particular connectors, we propose a way to easily model, study and compare different arbitration protocols.

3 Distributed model of multiple-bus architecture

Our purpose is first to provide a formal model for a multiple-bus architecture. Then, based on this model, different arbitration protocols could be easily modeled and analyzed. Using the notions of BIP framework already described in Section 2, we propose to model a multiple-bus architecture by defining the set of processors as a set of BIP atomic components. Comparing to the model presented in [9], we are here interested in multiple-bus and not a single one, thus, we use also a set of BIP atomic components to model the set of shared buses.

In the case of a centralized setting, a single centralized arbiter is modeled by a unique component and the arbitration protocol is carried out by the behavior of this unique arbiter. In this paper, as we are intending a distributed implementation of the bus allocation arbiter, no unique component is modeling the arbiter, but a set of *local* arbiters, each one associated to a component whether it is a processor or a bus. These local arbiters ensure arbitration by interacting with their peers. The set of local arbiters of the processors is, as the processors, modeled by a set of BIP components $\{P_1, P_2, \ldots, P_N\}$. The set of local arbiters associated to buses are also modeled by a set of BIP components $\{B_1, B_2, \ldots, B_M\}$ (see Figure 3). In our model, we do not explicitly model the different buses and processors as they are passive devices controlled each by a local arbiter. Thus, we only give the description of the different arbiters.

Arbitration protocols are then achieved by the interaction between the different arbiters. As already described in Section 2, interaction between components in BIP can be described in a compact manner by connectors (see Definition 3). In our model we describe how three well-known arbitration protocols could be modeled using a set of structured connectors.

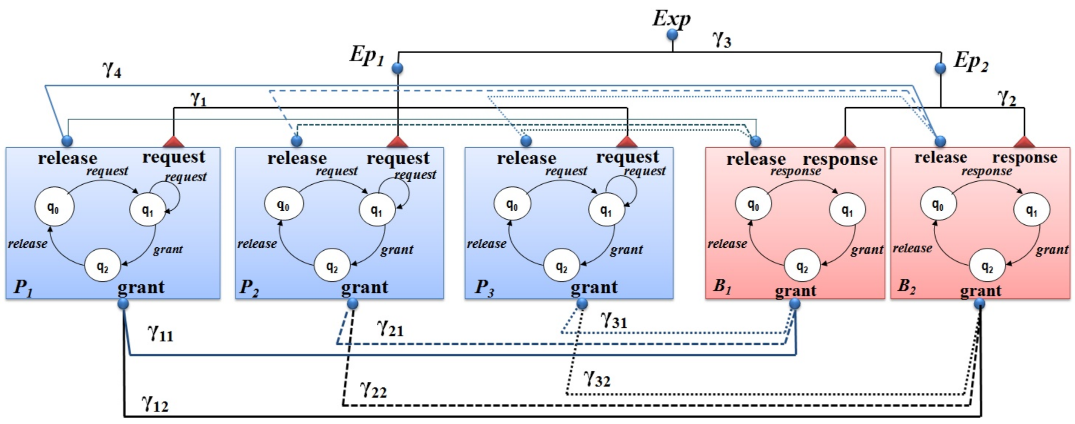

Fig. 3. A Distributed Model for multiple-bus architecture.

Our model contains N local arbiters associated each to a processor and M local arbiters associated each to a bus. For the sake of readability, we limit the figures describing our models to 3 processor arbiters namely $\{P_1, P_2, P_3\}$ and 2 bus arbiters namely B_1 and B_2 (see Figure 3). Note that, such a model can be easily extended to N and M arbiters (See Section 4).

Arbitration protocols are ensured by interactions between arbiters which are here the set of BIP components $\{P_1, P_2, P_3, B_1, B_2\}$. These interactions are encapsulated in a set of BIP connectors namely $\{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \{\gamma_{ij}\}_{i\in[1,3]}^{j\in[1,2]}\}$. In this work we focus on three well-known arbitration protocols namely: Fixed priority, Equal priority and Rotating priority protocols [2]. Figure 3 describes the overall structure of our model which is the same for the description of the different protocols. Indeed, the difference between the models of the different protocols is only some minor changes on connector characteristics. In other words, the behavior of components, and the structure of connectors are almost the same for all protocols. Each protocol is coded using variables, guards and functions associated to each connector (Definition 3). For this reason, we describe first the model, depicted in Figure 3, for the fixed priority protocol. Then we explain how it can be easily extended to model rotating and equal priority protocols. We start by the description of the behavior of the different BIP components, then we detail the set of connectors ensuring the bus arbitration protocol.

Figure 4 describes the behavior and the set of ports of BIP components modeling the processor arbiter P_i and the bus arbiter B_j. The two components have 3 states namely q_0, q_1 and q_2, where q_0 is the initial state. The different

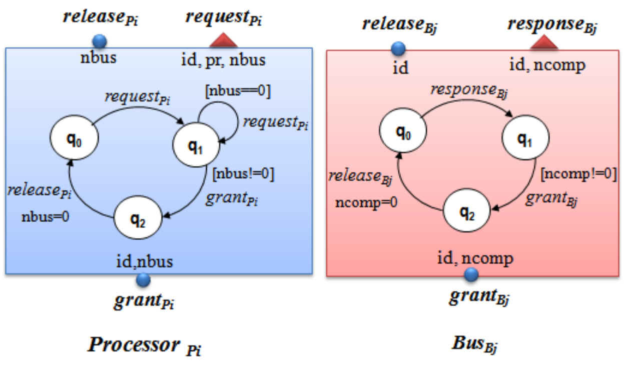

Fig. 4. Processor and Bus Arbiters.

transitions of the arbiters are labeled by one of their ports. These ports define how arbiters interact with the rest of components. The variables associated to these ports can be read and updated through connectors. The behavior of the processor arbiter can be described as follows:

- Initially in state q_0, P_i can make a request by firing the transition labeled by the port $request_{P_i}$. As this port is a trigger port, P_i can fire this transition with no need to synchronize with any other components.
- In q_1, two transitions are possible. If no bus is allocated, then P_i will fire continuously a loop transition $request_{P_i}$ until a $grant_{P_i}$ transition is performed. This transition labeled by $grant_{P_i}$ is fired when P_i gets a bus access. If it is the case, P_i goes to state q_2. As $grant_{P_i}$ is a synchron port, firing the corresponding transition depends on other conditions defined by the connectors connecting this port. These conditions will be detailed at the level of connectors description.
- In state q_2, P_i has already a bus access. It can stay in this state as much as it needs the bus. Once it decides to release it, the transition labeled by $release_{P_i}$ is then fired.

Figure 4 describes also the BIP component B_j of a bus arbiter. This arbiter is quite a passive arbiter, as the protocol policy is ensured by BIP connectors. Indeed, the bus arbiter receives the order to be assigned to a given processor through its $grant_{B_j}$ port. It, then receives the order to be released through its $release_{B_j}$ port. Details about how these orders are computed and sent are given in the description of connectors.

Fixed-Priority Protocol In this section, a detailed description of each connector is given. We first detail these connectors to model the fixed priority protocol. For the rest of protocols namely rotating and equal priority, we only give the updates to perform on those connectors. In fixed priority protocol, each processor is assigned a fixed and unique priority [5]. When more than M processors are requesting the bus at the same time, the M processors with the highest priority will get a bus. The policy of this protocol is mainly ensured by connectors depicted in

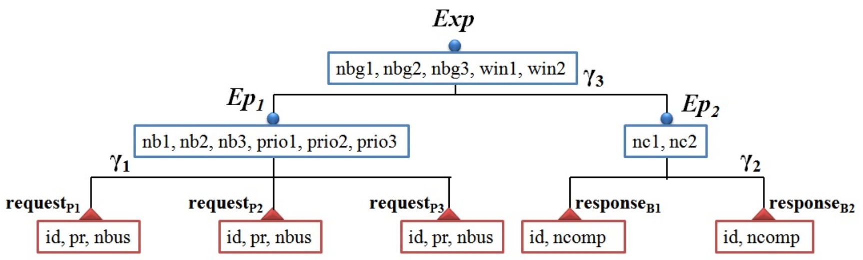

Fig. 5. A Structured BIP connector modeling arbitration protocols.

Figure 3. The set of connectors γ_1,γ_2 and γ_3 are detailed in Figure 5. The connector γ_1 is a broadcast connector which connects 3 ports ($request_{P_i}$, $\{i = 1, 2, 3\}$), one of each processor arbiter. It describes the request procedure by the set of its feasible interactions. In fact, as γ_1 is a broadcast connector, the set of its feasible interactions contains all possible subsets of its ports. The set of feasible interactions of γ_1 is $\{\{request_{P_1}\}, \{request_{P_2}\}, \{request_{P_3}\}$, $\{request_{P_1}, request_{P_2}\}$, $\{request_{P_1}, request_{P_3}\}$, $\{request_{P_2}, request_{P_3}\}$, $\{request_{P_1}, request_{P_2}, request_{P_3}\}\}$. In the Upward function of γ_1, collects the identifiers id and priorities pr of the processors asking for a bus access. The values of these variables will be then transferred to the exported port Ep_1, which exports this data to an upper connector namely γ_3.
The Downward function of γ_1, allows Ep_1 to send back, through the variable $nbus$, the id of the allocated bus. If no bus is allocated to P_i, then the corresponding $nbus$ will be set to 0.

Notice that, for a given interaction, the temporary values of connector variables when executing the down instructions are computed by the corresponding up instructions. However, these values are not accessible between different executions of the same interaction or between the execution of the transfer functions of different interactions. They are only stored between the execution of up and down instructions of the same interaction. In the other words, the temporary stored value is only accessible during the associated interaction and it will be lost when the interaction is achieved. This means that connectors are memoryless and no data can be stored at the connector level. This is a very interesting aspect which makes our model indeed distributed because connectors are completely different from components and they do not have static variables. On the other side, such a characteristic of connectors does not allow as to keep some information like the id of asking and not served processors. For this reason, we have modeled a loop transition labeled by the port $request_{P_i}$ which will be fired if no bus is allocated to P_i. In this way, the identifiers id and priorities pr of processors asking for a bus access can be collected again through the γ_1 connector at the level of the upward function. The presence of such a loop transition guarantees the no-blocking of our model after a first allocation cycle of buses.

γ_2 is connecting bus arbiters through the ports $response_{B_1}$ and $response_{B_2}$. By the means of its upward function, γ_2 exports to port Ep_2 the id of the available buses. The set of feasible interactions of γ_2 is $\{\{response_{B_1}\}, \{response_{B_2}\}, \{response_{B_1}, response_{B_2}\}\}$.

The rendez-vous connector γ_3 (see Figure 5), connects ports Ep_1 and Ep_2 which are respectively the exported ports of γ_1 and γ_2. When the guard $G = (Ep_1 \cdot nb_1 \neq 0 \vee Ep_1 \cdot nb_2 \neq 0 \vee Ep_1 \cdot nb_3 \neq 0) \wedge (Ep_2 \cdot nc_1 \neq 0 \vee Ep_2 \cdot nc_2 \neq 0)$ is provided, the interaction involving the ports Ep_1 and Ep_2 occurs. The guard G consists that at least there is one free bus and one processor which asks for the access to a bus.

This connector implements with its upward function, called the $DECIS$ algorithm (see Algorithm 1), the protocol policy to manage the allocation of buses. At the level of Ep_1, the id and priorities pr of the processors requesting a bus are stored and at the level of Ep_2, variables save the id of available buses. Thus, at the level of the exported port Exp of γ_3, all data needed to ensure the arbitration protocol is available.
Our proposed $DECIS$ algorithm consists in the selection of free buses as well as processors asking for the bus access, then according to the order of priority defined between processors the decision of the winners will be established. Indeed, the proposed algorithm consists in the following steps:

1. Create an array of structure named $Proc$ which has two members: id (identifier) and pr (priority). $Proc$ contains only the parameters of processors asking for the bus access.
2. Create an array named Bus which contains only the identifiers of free buses.
3. Call the function **SORT** that performs the sort of $Proc$ according to the priority pr.
4. Assign for each free bus the identifier of the winner processor and for each processor the identifier of the allocated bus.

Algorithm 1 DECIS: For N processors and M buses

```
Require: interaction $\{Ep_1, Ep_2\}$
  Input: int $nb[N]$, $Prio[N]$, $nc[M]$
  Output: int $win[M]$, $nbg[N]$
  $j \leftarrow 1$
  for $i := 1; i <= N; i++$ do
    if $nb[i] \neq 0$ and $Prio[i] \neq 0$ then
      $Proc[j].id \leftarrow nb[i]$
      $Proc[j].pr \leftarrow Prio[i]$
      $j \leftarrow j + 1$
    end if
  end for
  $k \leftarrow j$
  $j \leftarrow 1$
  for $i := 1; i <= M; i++$ do
    if $nc[i] \neq 0$ then
      $Bus[j] \leftarrow nc[i]$
      $j \leftarrow j + 1$
    end if
  end for
  $l \leftarrow j$
  SORT$(Proc, k)$
  for $i := 1; i <= l; i++$ do
    $idB \leftarrow Bus[i]$
    $idP \leftarrow Proc[i].id$
    $win[idB] \leftarrow idP$
    $nbg[idP] \leftarrow idB$
  end for
```

In the downward function of γ_3, the different variables are updated so that each processor gets the id of the allocated bus (through port Ep_1) and each bus gets the id of the processor for which it was allocated (through port Ep_2).

Notice that, if one decides to model a different protocol, the main modifications will be at the level of this connector.

As soon as, processors get the *id* of the allocated bus (through γ_1) and buses receive the *id* of the assigned processor (through γ_2), transitions labeled by ports $grant_{P_i}$ and $grant_{B_i}$ are fired and the interaction $G_i = \{grant_{P_i}, grant_{B_j}\}$ can take place. This interaction is achieved through the γ_{ij} connector (see Figure 6).

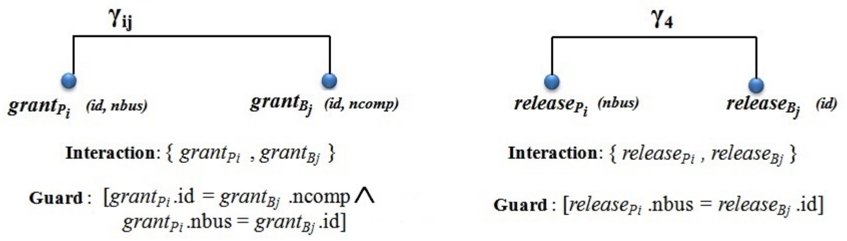
γ_{ij}
$grant_{P_i}$ (id, nbus)
$grant_{B_j}$ (id, ncomp)
Interaction: { $grant_{Pi}$, $grant_{Bj}$ }
Guard: [$grant_{Pi}$.id = $grant_{Bj}$ .ncomp ∧ $grant_{Pi}$.nbus = $grant_{Bj}$.id]
γ_4
$release_{P_i}$ (nbus)
$release_{B_j}$ (id)
Interaction: { $release_{Pi}$, $release_{Bj}$ }
Guard: [$release_{Pi}$ .nbus = $release_{Bj}$ .id]

Fig. 6. Grant connector γ_{ij} and Release connector γ_4.

Later, when a processor P_i decides to release a bus B_j, then the transition $release_{P_i}$ is fired and the interaction $\{release_{P_i}, release_{B_j}\}$ took place. This interaction is performed through γ_4 connector (see Figure 6).

Rotating-Priority Protocol In the rotating priority protocol, each device is given a different priority in each arbitration cycle [5,14]. Which means that, if a device has the highest priority in a first cycle, it will have the lowest priority in the next one, if it gets a bus. Priorities in this protocol are not static as they are updated after every bus allocation. To model this protocol, we propose to build upon the already described model. The idea is simply to add a new port $update_{P_i}$ for each processor arbiter (see Figure 7). This port is connected to *grant* port of each bus arbiter which allows to update the priority once a bus allocation is performed (see Figure 7). The transition labeled by the port $update_{P_i}$ is then possible in all processor states. Thanks to this new connector, no changes in other existing connectors are needed as each priority variable *pr* will always take the updated value.

Equal-Priority Protocol In the equal priority protocol, all devices have equal priority of getting a bus upon request. When the number of requests is less or equal to the number of free buses, all requests will be granted. If the number of requests is higher than the number of free buses, then the choice of the winners depends in general on the priority assigned to processors. However, when no priority is defined between processors, which is the case of the equal priority protocol, any conflicts between requests to the buses will be solved at the level of the BIP engine using the FIFO (First In Fist Out) rule. In fact, to model the equal priority protocol, we use the same structure as the BIP model of the

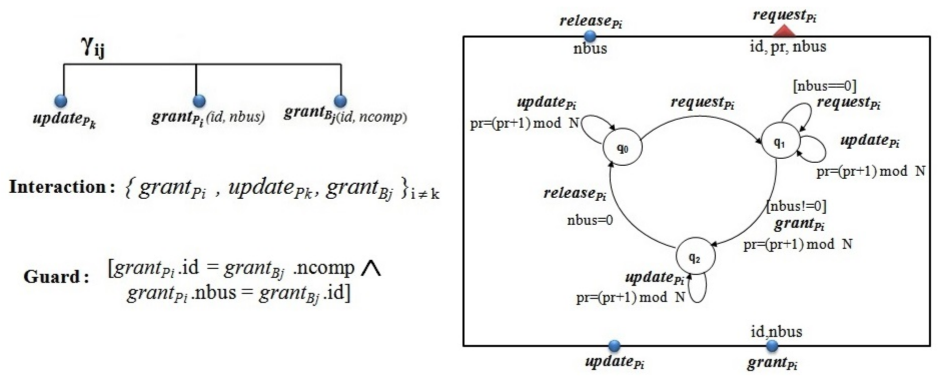

Fig. 7. Processor behavior and Update Connector for Rotating priority protocol.

fixed priority protocol with almost the same behavior and some modifications explained as follow:

- When the conflict between requests to the buses is managed using FIFO rule, there is no need to implement the $DECIS$ algorithm(see Algorithm 1) for this protocol. Then, no data transfer is needed at the level of the structured connector (see Figure 5), thus the upward and downward functions of this connector are removed. This structured connector allows so only the transfer of the requests of various processors to the available buses.

- When no priority is defined between processors, the variable pr associated to the port $request_{P_i}$, defining priority for each processor, is removed.
- At the level of the grant connector γ_{ij}, the guard is removed and the following affectations are added:

$$grant_{B_j}.ncomp \leftarrow grant_{P_i}.id \ ; \ grant_{P_i}.nbus \leftarrow grant_{B_j}.id$$

- At the level of the behaviors of both the bus and the processor controllers, the only modification is the elimination of the guards of the grant transitions.

4 Implementation and Experimental Results

We propose to use BIP tools to verify and analyze the different models proposed previously. In particular, we use the DFinder tool [15] offered by BIP toolbox to verify the property of deadlock freedom of the already described arbitration protocols. DFinder is a compositional verification tool for deadlock detection where verification is applied only to high level models for checking safety properties such as invariants and deadlock-freedom. Based on this tool, we have proven the deadlock freedom of the already described BIP models of the different arbitration protocols at this abstract level of our model and with no need to code generation. This is very interesting in the case of studying new protocols, one

Protocols	Equal Priority	Fixed Priority	Rotating Priority
Time(ms)	48	57	187

Table 1. Deadlock Verification Time.

can decide about deadlock freedom of any protocol at a high-level model and thus can modify and adjust the protocol easily.

We ran our experiments on a Linux machine with Intel(R) Core (TM)2 Duo $2.93 GHz \times 2$ and $16 GiB$ memory. Table 1, shows verification time taken by the tool for analyzing and detecting deadlock for our model. The time is computed for a model with 5 processors and 2 buses.

N × M (Nbr of interactions)	25×15 (754)	30×20 (1204)	75×25 (3754)	100×50 (10004)
Time(Second)	2.7	3.8	120.1	814.9

Table 2. Analyzing/Detecting Deadlocks for Equal Priority Protocol

Extending our model to any number of processors and buses could be easily performed as the different BIP components (processors and buses) have the same behavior. For instance, we measure, in Table 2, the verification time for the equal priority model for systems with more important number of components. Extending easily the model to an important number of processors and buses provides a way to study how protocols may react in the context of more complex architectures and thus with more conflicts to manage. Notice that increasing the number of components increase consequently the number of conflicts interactions and so require more a more important time of exploration to detect deadlocks and more memory resources.

$k \times 10^2$	Equal-Priority					Fixed-Priority					Rotating-Priority				
	P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5	P_1	P_2	P_3	P_4	P_5
1	22	21	20	18	19	47	50	0	0	0	19	20	21	22	18
10	196	215	192	191	206	491	506	0	0	0	198	201	202	203	196
100	2013	1997	2007	1990	1993	4912	5085	0	0	0	1997	2002	2005	1994	2002

Table 3. Model Based on Connectors: Number of grants per processor ($k \times 10^2$ total requests)

Once a model is proven to be deadlock free, further performance analysis could be provided based on the code generated from BIP models. We now give some experimental results obtained through this automatically generated code to study fairness property of the different protocols. Indeed, fairness is a key property one it comes to allocation protocols [16]. Ensuring fairness by a protocol means that all processors will eventually get a bus if they ask for it. Table 3 gives the number of times a processor gets the access to a bus for an architecture of 5 processors and 2 buses.
In the case of the Fixed priority protocols we suppose that priorities are defined

as follows: $P_5 < P_4 < P_3 < P_2 < P_1$. As expected, the fixed priority protocol does not ensure fairness. The processors with the lower priorities (in this case P_3, P_4 and P_5) never get a bus access, if the rest of processors with higher priorities ask for a bus. For this reason, the total of grants of different processors is equal to the number of requests minus three, that is the three requests of the three processors with the lower priorities whose never get a bus access.

The equal and the rotating priority protocols indeed ensure fairness as all processors get an access to a bus in almost the same proportion. The equal priority protocol is not always expected to ensure fairness as it depends on the implementation of the required arbitration mechanism. If it implements fair choice then fairness is guaranteed by the protocol. Notice that, in the results presented in Table 3, our equal priority model is fair. This is expected as, in the absence of specified priorities, the mechanism implemented in BIP treats requests in a FIFO order. This means that any processor that have made a request will eventually get a bus access. We propose also to check the mutual exclusion on bus access for an architecture of 3 processors and 2 buses. The scenario where two or more processors get access to the same bus at the same time can never take place. Indeed, in our model depicted in Figure 5 the grant port of each bus component $\{B_j\}_{j\in\{1,2\}}$ is connected to three different rendez-vous connectors $\{\gamma_{1j}, \gamma_{2j}, \gamma_{3j}\}_{j\in\{1,2\}}$. According to BIP semantics [17], it is impossible to fire two synchronization involving common ports. This ensures that a bus can be granted for one processor at a time.
The use of BIP is also highly motivated by the fact that BIP tools [18] offer an automatically generated distributed code. Thus, once a model is proven to be deadlock free or a set of relevant properties have been analyzed (such as fairness), further performance analysis could be provided based on the code generated [19].

5 Related Work

Previous researches as in [4,2,3] have been interested by the design of arbitration protocols for multiple-bus architecture. To analyze, evaluate and compare the performance of different arbitration protocols, they have based only on estimation (probabilistic models [4,3] and analytical models [2]). Thus, the obtained results remain rough and thus not safe because they are based only on simulations. As the choice of the adequate arbitration protocol is critical and has a direct effect on the performance of the system, we thus need a powerful method which provides correct and fully reliable results. For this reason, we have opted for formal methods which are a particular kind of mathematically based techniques for the specification, development and verification of systems. The use of formal methods for system design provides reliability, correctness and robustness of a design. In our research, we choose to describe the multiple bus architecture using a component based framework BIP (see Section 2) which provides correctness by construction and tools for formal verification of systems properties [7,19]. We were able to design a formal model which specifies in an elegant way the multiple bus system for three priority schemes (equal priority, fixed priority and rotating priority). The major advantage of our model is that it is almost the same for the three policies with minor changes at the level of the proposed algorithm which manages conflicts between concurrent processors. Unlike the previous works that propose a distinct model for each arbitration protocol. Moreover, we have performed a formal validation of our model by verifying deadlock freedom. We have also verified the fairness property for each protocol which allows to compare the performance of different policies and so decide the most adequate for multiple bus system. In [9], a formal model have been also proposed using the BIP framework but only for a single bus architecture which makes conflict management completely different from our case. Moreover, in multiple-bus architecture the distributed issue is more relevant than in [9]. Note that in the different previously preformed researches, the proposed models are only used for analysis and performance comparison and no code generation is offered. Unlike ours, which in addition to formal validation, provides a distributed automatically generated code [20].

6 Conclusion

We have proposed a high-level formal and abstract model for multiple-bus multiprocessors architecture. The proposed model provides a way for describing different arbitration protocols in a distributed and abstract manner, which allows to easily analyze and verify their different properties without having to implement them in a concrete system. Indeed, in this paper we have proposed several models for three well-known protocols namely, *equal priority*, *fixed priority* and *rotating priority*. We have also performed several property verification of the studied protocols and derived a distributed implementation of the proposed models. Then experimental results have been provided using the different tools provided by the BIP framework. Different new protocols could be also easily described using the same principle, in particular if it is based on the same multiple bus architecture. As future work, we are considering to model new SoC communication architectures as crossbar architecture [21] and thus to study and verify its corresponding arbitration protocols as Round-Robin Protocol [22] and Time Division Multiple Access (TDMA) [23].

References

1. N.Doifode, D., Dr.P.R.Bajaj: Design and performance analysis of efficient bus arbitration schemes for on-chip shared bus multi-processor soc. International Journal of computer Science and Network Security (IJCSNS) **8**(9) (2008)
2. C.-M. Chung, D.A.C., Q.Yang: A comparative analysis of different arbitration protocols for multiprocessors. Journal of Computer Science and Technology **11**(3) (1996)
3. John, L.K., Liu, Y.: Performance model for a prioritized multiple-bus multiprocessor system. IEEE Trans. Computers **45**(5) (1996) 580–588
4. Yang, Q., Raja, R.: Design and analysis of multiple-bus arbiters with different priority schemes. In: Parallel Architectures (Postconference PARBASE-90). (1990) 276–295

5. El-Guibaly, F.: Design and analysis of arbitration protocols. IEEE Trans. Comput. **38** (1989) 161–171
6. J.-P. hilippe Fassino, J.-B Stefani, J.L., Muller, G.: Think: A software framework for component-based operating system kernels. 16th IEEE Real Time Systems Symposium (2002)
7. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: SEFM, IEEE Computer Society (2006) 3–12
8. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A.L.: Metropolis: An integrated electronic system design environment. IEEE Computer **36**(4) (2003) 45–52
9. I.Ben-Hafaiedh, S., M.Jaber: Model-based design and distributed implementation of bus arbiter for multiprocessors, IEEE ICECS (2011) 65–68
10. Graf, S., Quinton, S.: Contracts for BIP: hierarchical interaction models for compositional verification. In: Proc. of FORTE'07. Volume 4574 of LNCS. (2007) 1–18
11. Bliudze, S., Sifakis, J.: The algebra of connectors - structuring interaction in bip. IEEE Trans. Computers **57**(10) (2008)
12. Bozga, M., Jaber, M., Sifakis, J.: Source-to-source architecture transformation for performance optimization in BIP. In: Proc. of SIES'09. (2009) 152–160
13. Basu, A., Bidinger, P., Bozga, M., Sifakis, J.: Distributed semantics and implementation for systems with interaction and priority. In: Proc. of FORTE'08. (2008) 116–133
14. Kulmala, A., Salminen, E., Hamalainen, T.: Distributed bus arbitration algorithm comparison on fpga based mpeg-4 multiprocessor soc. In: Norchip Conference, 2006. 24th. (nov. 2006) 167 –170
15. S.Bensalem, M.Bozga, P.H.N., J.Sifakis: D-finder: A tool for compositional deadlock detection and verification. In: Computer Aided Verification. (2009) 614 – 619
16. Taub, D.: Arbitration and control acquisition in the proposed ieee 896 futurebus. IEEE Micro **4** (1984) 28–41
17. Bliudze, S., Sifakis, J.: Algebraic semantics of hierarchical connectors in the BIP framework. Techreport, verimag (February 2007)
18. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: A framework for automated distributed implementation of component-based models. Distributed Computing **25** (2012) 383–409
19. Bliudze, S., Cimatti, A., Jaber, M., Mover, S., Roveri, M., Saab, W., Wang, Q.: Formal verification of infinite-state BIP models. In: Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings. (2015) 326–343
20. Bozga, M., Jaber, M., Sifakis, J.: Source-to-source architecture transformation for performance optimization in bip. In: SIES, IEEE (2009) 152–160
21. S.Murali, L., Micheli, G.: An application-specic design methodology for on-chip crossbar generation. Computer-Aided Design of Integrated Circuits and Systems **26**(7) (2007) 1283–1296
22. E.S.Shin, V.J. Mooney, G.: Round robin arbiter design and generation, International Symposium on system synthesis (2002)
23. K. Lahiri, A. Raghunathan, G.L.: Lotterybus: A new high performance communication architecture for system-onchip designs, Design Automation Conference (2001) 15–20