Model-based design and Distributed Implementation of Bus Arbiter for Multiprocessors

Imene Ben-Hafaiedh
VERIMAG/UJF
2, Avenue de Vignate
38610 GIERES, FRANCE
Email: Imen.Ben-Hfaiedh@imag.fr

Susanne Graf
VERIMAG/CNRS
2, Avenue de Vignate
38610 GIERES, FRANCE
Email: Susanne.Graf@imag.fr

Mohamad Jaber
VERIMAG/UJF
2, Avenue de Vignate
38610 GIERES, FRANCE
Email: Mohamad.Jaber@imag.fr

Abstract—The contribution of this paper is twofold. First we propose a high-level distributed and abstract model of the bus arbiter for multiprocessors. Our model provides a way for describing several existing arbitration protocols in a distributed and abstract manner so that their properties and performance could be easily compared and analyzed. Second, we propose to automatically verify deadlock freedom property of these protocols and to automatically generate their distributed implementation.

I. INTRODUCTION

In most common digital systems (SoC), a set of independent devices (processors, memories, links ...) share a common bus. The access to this shared resource is in general managed by a bus arbiter deciding which device can access the bus and when. The performance of these systems depends considerably on the protocol used by the arbiter to bus allocation and how this protocol could be balanced and time efficient. Thus, the arbitration protocol plays a crucial role in determining the performance of bus based system as it assigns the priorities with which device is granted the access to the shared communication resources.

High-level system performance could be ensured by choosing simple and fast arbitration protocols [12], [15] and analyzing properties of these different protocols at a high-level system description could be considerably helpful to compare their performance without having to go into a very low-level implementation details. We argue that arbitration protocols may alternatively be specified in terms of rich communication between devices through high-level features such as multi-party interactions and global priorities. Often, using these concepts leads to much more concise specifications which are more adequate to provide an understanding of the global behavior of the protocol and for the verification of the global properties. Our first aim is to specify the bus arbitration problem for multiprocessors at a high level of abstraction so that we can study different properties and issues of such a problem. Moreover, we propose to model well-known arbitration protocols so as we can easily compare and analyze them without having to go further into their implementation details [7], [10]. In most arbitration protocols, two issues are interesting to be studied namely fairness and priorities [13]. Priority is assigned to each processor connected to the bus, so that we can decide which one will have the access when more than one processor are requesting the bus. Fairness [7] guarantees that the processor with lower priority will eventually have the bus access. Once arbitration protocols have been described, their implementation could be achieved in a centralized or a distributed way. Several studies have been carried out based on a centralized arbitration approach [5], [11]. However, the distribution of arbiters is necessary in fault-tolerant system where a failure of a specific arbiter should not cause the fail of the complete system. Nonetheless, modeling and implementing distributed arbiter protocols is time consuming and error-prone. In this paper, we propose to derive automatically correct and distributed implementation [3], [4] from the high level description of the arbitration protocols. In [8], they have proposed a distributed implementation but for a particular arbitration protocol. Our approach is more general as it allows to derive distributed implementation from any arbitration protocol once it is specified using the high-level model we propose.

Paper Organization: In Section II, we present the different ingredients needed to define a high-level description model of the arbitration problem. Then, in Section III, we describe how to use these ingredients to define abstract and distributed models of several well-known arbitration protocols. We also describe how these protocols could be verified based on these models. Section IV depicts some experimental results obtained from our automatically derived implementation. Section V concludes and hints for future work.

II. BUS NETWORK ARCHITECTURE MODELS

In this section, we present a high-level modeling formalism for the description a bus network abstract model. We choose to specify our architecture using the BIP (Behavior-Interaction-Priority) component framework [1], [4] as it is a framework with formal semantics that rely on multi-party interactions for synchronizing components and priorities for scheduling between interactions. Moreover, this framework provides different tools for property verification and code generation.

Definition 1 (Atomic component): An atomic component $K$ is a Labeled Transition System (LTS) defined by a tuple $(Q, P, \delta)$ where $Q$ is a set of states, $P$ is a set of communication ports and a transition relation $\delta \subseteq Q \times P \times Q$.
As usually, \( q_1 \xrightarrow{a} q_2 \) denotes \( (q_1, a, q_2) \in \delta \). In practice, atomic components could be extended with variables. 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 \( K_i = (Q_i, P_i, \delta_i) \) 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. Where 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.

The composition of BIP components with a set of interactions \( \gamma \) defines an LTS (Definition 3).

**Definition 3 (Composition of components):** The composition of \( n \) components \( K_i \) with a set of interactions \( \gamma \subseteq 2^P \) is denoted by \( K = \gamma(K_1, \ldots, K_n) \) and it defines an LTS \( (Q, P, \delta) \) such that \( Q = \prod_{i=1}^{n} Q_i \) and \( \delta \) is the least set of transitions satisfying the following rule:

\[
\begin{align*}
\alpha = & \{p_i\}_{i \in I}, \forall i \in I. q_1^i \xrightarrow{P_i} q_2^i \land \forall i \not\in I. q_1^i = q_2^i \land \\
& (q_1^1, \ldots, q_1^n) \xrightarrow{\alpha} (q_2^1, \ldots, q_2^n)
\end{align*}
\]

Our goal is to specify the different arbitration protocols at high-level and distributed way so that we can easily analyze them. The choice of BIP for this purpose is highly motivated by the facility of defining a large spectrum of compositions by means of interactions models and also priorities amongst interactions, defined hereafter. Indeed, BIP allows also to easily define priority which, as already stated, is an important issue when dealing with arbitration protocols.

**Definition 4 (Priority in BIP):** A priority order denoted by \(<\) is a strict partial order on the set of interactions \( \gamma \). We denote that an interaction \( a \) has lower priority than \( b \) by \( a < b \). A system \( S = (Q, P, \delta) \) controlled by a priority order \( < \) defines an LTS \( (Q, P, \delta_<) \) where \( \delta_< \) is defined by the following rule:

\[
q_1 \xrightarrow{a} q_2 \land \exists b \in \gamma. (a < b \land q_1 \xrightarrow{b} q_2)
\]

Using these notions, a bus network architecture is easily modeled by defining the set of processors as a set of BIP atomic components \( \{P_1, P_2, \ldots, P_N\} \). As usually, we do not explicitly model the bus device as it is a passive device controlled by the bus arbiter.

In the case of a centralized setting, the centralized bus arbiter is also modeled by a unique atomic component. In a centralized setting the arbitration protocol is carried out by the bus arbiter behavior. As we are intending a distributed implementation of the bus arbiter, no unique component is modeling the arbiter, but a set of local arbiters, each one associated to a processor and ensuring arbitration by interacting with its peers and giving its processor access to the bus when the set of arbiters agree on that. Similarly, these local arbiters are also modeled by a set of BIP components \( \{A_1, A_2, \ldots, A_N\} \). The arbitration protocol is ensured by communication interaction amongst these components \( \{A_1, A_2, \ldots, A_N\} \). Such communications are modeled by BIP interactions (see Definition 2). Arbitration protocols can be easily achieved by means of BIP interactions between local arbiter components and priority between the corresponding interactions. In the next Section, we provide such descriptions for several well-known arbitration protocols.

### III. Distributed Abstract Models of Arbitration Protocols

Based on the already defined bus network architecture model, we present how one can describe, in an abstract manner, different well-known arbitration protocols. In BIP, we represent a protocol in a very abstract, declarative manner, as a set of allowed transitions by defining the set of allowed interactions which guarantee that at most one \( P_i \) of the set of processors components \( \{P_1, P_2, \ldots, P_N\} \) accesses the Bus. How these joint transitions are achieved later by messages exchanged amongst the set of arbiters \( \{A_1, A_2, \ldots, A_N\} \) is not part of our model, and the distributed protocol is obtained later automatically by an appropriate code generator [3].

In the following, we show that using almost the same abstract model one can describe easily different arbitration protocols without having to make any modifications in the behaviors of arbiters components but only by defining a new set of interactions and priorities between them. We also propose to analyze and verify arbitration protocols using BIP verification tools.

In this paper, we focus on three arbitration protocols that are commonly deployed in actual systems, namely equal-priority protocol, fixed-priority protocol and rotating-priority protocol.

#### A. Equal-Priority Protocol

In the equal-priority protocol all processors have equal priority for accessing to the bus. Thus some mechanism is needed to deal with conflicts which may occur when several processors request the bus at the same time. Such a mechanism is discussed in [9]. The BIP model of this protocol is depicted in Figure 1-a, where we provide the description of a network of 3 processors and which can be naturally extended to \( N \) processors. Each processor component has 3 ports, namely \( request_{P}, grant_{P} \) and \( release_{P} \) allowing it to interact with its local arbiter \( A_i \). The interaction \( \{request_{A_{i}} , request_{P_{i}}\} \) takes place when \( P_i \) requests the bus access. Similarly, releasing the bus after completing a current transaction, is also modeled by an interaction between both components namely \( \{release_{A_{i}} , release_{P_{i}}\} \). The behavior of an arbiter \( A_i \) is detailed in Figure 1-a. It has 4 ports namely \( request_{A_{i}} , grant_{A_{i}} , release_{A_{i}} \) and \( accept_{A_{i}} \). In state \( q_0 \), \( A_i \) has not received yet any request from \( P_i \) and it can perform the transition labeled by the port \( accept_{A_{i}} \). This port is involved
in a set of interactions allowing other processors to get the bus access. Once in state $q_1$, $A_i$ has already performed the interaction $\{request_{A_i}, request_{P_j}\}$, means that it has received a request from $P_j$ and thus it can try to get the bus by performing the transition labeled by the port $grant_{A_i}$ involved in the interaction $G_i = \{grant_{A_i}, grant_{P_j}, \{accept_{A_i}\}_{j \neq i}\}$. In the same state, $A_i$ allows also other processors to get the bus access by performing any of the interactions $G_{j\neq i} = \{grant_{A_i}, grant_{P_j}, \{accept_{A_i}\}_{j \neq i}\}$. Note that, there is a state of the model in which the interactions $G_1, G_2$ and $G_3$ are possible. Different mechanisms can be proposed to deal with such a conflict and the mechanism provided by BIP gives the bus access to the first processor requesting it.

**B. Fixed-Priority Protocol**

In the fixed-priority protocol each processor is assigned a unique and fixed priority. When two or more processors are requesting the bus access at the same time, the processor with the highest priority will be the winner. Examples of this protocol are presented in [6], [14]. Note that this protocol does not ensure fairness as the processor with the highest priority can always get the bus. This protocol is naturally modeled using the same atomic arbiter components, the same global interaction model (as mutual exclusion needs still to be ensured) but by defining now a priority between interactions $G_i$ (see Figure 1). The idea consists of simply defining a priority between the $G_i$ interactions of the arbiters as BIP models provides a way to naturally define priority between interactions. For example, defining the following priority rule: $G_3 < G_2 < G_1$ gives the priority to the processor $P_1$ over the rest of processors and thus whenever this processor wants to perform the interaction $G_1$ when the bus is available, it will get it (see Figure 1-b). Moreover, we may modify at any time the priority between processors without having to modify the network architecture or any details of the devices.

**C. Rotating-Priority Protocol**

An example of the rotating-priority protocol is provided in [7]. In such a protocol, priority is not fixed which means that the processor having the highest priority, will loose it once it gets the bus access. Unlike the previously described protocols, this protocol ensures fairness between processors. Indeed each of the processors will get highest priority if it requests the bus access. To model this protocol, we propose again to build upon the same BIP architecture by just extending the interface of the arbiters with some new integer variable $x_i$ used to manage the dynamic priority: starting from a state in which all values $x_i$ are distinct, at each interaction $G_i$, all values $x_i$ are increased by 1 modulo $N$. At any time, $G_i$ is defined to have higher priority than $G_j$ if $x_i > x_j$ (see Figure 1-c). For instance, at the beginning the variable $x_N$ of arbiter $A_N$ is equal to $N-1$ and $A_N$ has the highest priority. So, if it requests the bus access, it will get it by performing the interaction $G_N$ and all variables of all arbiters are updated to $(x_k + 1) \mod N$. Which means that $x_N$ will be updated to 0 and $A_N$ will loose its highest priority. However, if $A_N$ does not request the bus, the priority goes to the arbiter whose variable is equal to $N-2$. Note that with such priority rules we always give the priority to the arbiter who has been waiting the longest time. This shows the high modularity of BIP. We do not need to modify the behavior of the arbiter components (this could be external code) but only the interfaces and the interaction model which are in the hands of the system integrator.

**Analysis and Verification**

We have proposed an abstract model for each of the studied protocols based on the same basic set of components. In fact, describing a new protocol requires only modifying the set of interactions and priorities while keeping the same behaviors of components. We propose to use BIP tools to verify and analyze these protocols, in particular, we applied the deadlock finder tool [2] provided by the BIP tool chain. This tool computes sufficient conditions that guarantee deadlock freedom, and if necessary allows to refine them successively until either deadlock freedom is proved or an actual deadlocking execution is found. As we have algorithms which allow to derive from any deadlock free BIP specification a deadlock free distributed specification (where communication is by message passing), this means that we can verify and solve deadlock freedom at the abstract BIP level, and do not need to do this for the lower level specification where interaction between arbiters is exclusively via binary message passing and which is much more complex to verify. Table I shows the execution time for analyzing and detecting deadlock by increasing the number of arbiters as well as processors. For instance, analyzing a model consisting of 300 components (150 arbiters and 150
processors) requires only 30 seconds to be checked. At the level of the detailed message passing protocol, verification of deadlock freedom would have been far too complex.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

It was stated in the previous section that arbitration protocols can be easily analyzed and verified using BIP verification tools. However, further performance analysis could be provided once these protocols have been implemented. We now present some experimental results obtained through automatic code generation from the previously described BIP models. Using the tool presented in [3], completely distributed implementations of arbitration protocols models can be derived.

Tables II and III present the results related to the fairness of the different protocols. They give the number of times a processor gets the access to the bus computed, in Table II, for an architecture of 3 processors (3 arbiters) and by increasing the total requests to the bus \((k \times 10^2)\). In Table III, we consider an architecture of 8 processors (and thus 8 arbiters) for each protocol where the total number of requests is \(10^3\).

As expected, the fixed priority protocol does not ensure fairness, as for example \(P_1\) which has highest priority gets considerably more bus accesses than all other processors. The non prioritized version of the protocol seems to work, nevertheless this result is likely to be quite "implementation dependent". However, the rotating priority protocol indeed ensures fairness as all processors access to the bus in almost the same proportion. Notice that, in our experiments we have only considered the case where all processors require bus access all the time. For other request profiles, the results will be different. E.g. the fixed priority protocol is likely to be appropriate if the request frequency is inversely proportional to the priority (the higher the priority, the lower the request rate).

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 Tables II and III, 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 the bus access.

V. CONCLUSION

We have proposed an abstract model for describing arbitration protocols. Such a model provides a way to easily analyze and verify such protocols without having to implement them in a concrete SOC. Moreover, our model is rich enough so that it ensures a relatively simple and abstract way for describing different protocols. Indeed, in this paper we have proposed models for three well-known protocols, however different new protocols could be also easily described using the same principle. Our approach is highly motivated by the different analysis and verification tools provided by BIP framework. In particular, the tool offering to automatically derive distributed implementation of these protocols. For future work, we are considering new arbitration protocols to be modeled and analyzed. We are also intending to use the same approach to multiple bus arbitration problem, which leads to additional conflict situations to handle between processors and buses.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>(P_1)</th>
<th>(P_2)</th>
<th>(P_3)</th>
<th>(P_4)</th>
<th>(P_5)</th>
<th>(P_6)</th>
<th>(P_7)</th>
<th>(P_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal-Priority</td>
<td>114</td>
<td>117</td>
<td>133</td>
<td>124</td>
<td>135</td>
<td>117</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>Fixed-Priority</td>
<td>596</td>
<td>327</td>
<td>70</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rotating-Priority</td>
<td>123</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

TABLE III

NUMBER OF GRANTS PER PROCESSOR FOR 1000 REQUESTS TO THE BUS.

REFERENCES