Formal Verification of an Arbiter Cascade

Hartmann J. Genrich
Gesellschaft für Mathematik und Datenverarbeitung
Schloß Birlinghoven, 5205 Sankt Augustin 1, Germany

Robert M. Shapiro
Meta Software Corporation
125 CambridgePark Dr, Cambridge, MA 02140, USA

Abstract

The asynchronous access of a group of users (e.g. processors) to a single resource (e.g. bus) is regulated by a cascade of arbiters. A single arbiter circuit handles two users. The cascade permits any number of users to be serviced. We use a hierarchical Colored Petri Net to describe the arbiter circuit and the protocol for using it. We also describe the layout of a 2d input cascade of (2d-1) arbiters, d ≥ 1 being the depth of the cascade. We verify the proper functioning of the cascade, first for depth d = 1 using an occurrence graph analyzer to prove crucial invariants and conformance to the protocol; then for arbitrary depth using mathematical induction. As an alternative proof, we develop equivalent Petri net substitutes for the building blocks of the design and verify the resultant special net using classical net theoretic methods. Based on the verification we propose a change of the arbiter to speed-up the cascade.

1 The Task

Figure 1 shows a cascade of depth d = 2 with 3 arbiters serving 4 users. (The bus itself is not shown). The users may issue requests independently by setting their request-out line, r, to H(igh). A request is granted eventually by the corresponding ackn-in line, a, being set to H, the guarantee being that no other users' request and acknowledgement lines are H at the same time. Eventually the user releases the grant by setting its request-out to L(low). Before issuing another request it waits for the release to be acknowledged by its ackn-in going low.

Each arbiter serves two users at its input side and acts as a single user at its output side. Thus the task of an arbiter is to reduce the behavior of two independent users to the behavior of a single one. The cascade is terminated by a trivial user complement that converts requests into acknowledgements.

Figure 2 shows the first level of detail of the arbiter, an abstract view of the user, and the user complement (co-user). The switching elements used to model these components are elaborated in figure 3.
The model was built using Design/CPN [1]. A central feature of this hardware-oriented net model is that the (one-directional) lines are represented as boolean places that always carry exactly one token of alternating colors, named H and L. It is not the tokens but the changes of their colors that flow through the net. On the lowest level of detail (see figure 3), the elements of the circuit are modeled as transitions that maintain the functional dependencies of the output values on the input values. The guards allow the transitions only to occur if at least one output value changes.

The model as presented here is adequate in so far as it correctly represents the functional relationships. However, it totally ignores the timing relationships due to switching and propagation delays. The circuit is not self-timed. More about this issue later. The model was carefully tested by means of interactive simulation and a series of test runs in compiled form. However, only formal verification could provide the expected degree of confidence.

For formulating the protocols to which the users and the arbiters have to conform we use the following notation. If x denotes a line, then x+ and x− denote its two states of being high and low, respectively. !x+ and !x− denote the acts of x being set to high respectively low at its initial end, ?x+ and ?x− denote the acts by which these changes on x are sensed, respectively processed, at the terminal end of x.

Hence, the protocol for a persistent use of line x is

\[ \@(!x+ \ ?x+ \ !x- \ ?x-) \]  

(L)

where \( \@ \) denotes the endless repetition of a sequence of acts.

(L) suggests that x is low initially. We will use a marker, •, to indicate a different state (initial or current). For example,

\[ \@(!x+ \ ?x+ \ •x- \ ?x-) \]  

(L′)

indicates that x is high but the last change to high has not been processed yet.

The user protocol, the simple continuous cycle of ("send a request, wait for a grant, send a release, wait for an OK) is formulated as

\[ \@(!r+ \ ?a+ \ !r- \ ?a-) \]  

(U)

and the state of the user when it has access to the bus is depicted by

\[ \@(!r+ \ ?a+ \ •r- \ ?a-) \]  

(U′)

The co-user protocol is the closure of (U), ! and ? being interchanged,

\[ \@(?r+ \ !a+ \ ?r- \ !a-) \]  

(B)

and finally, the arbiter protocol is

\[ \@(?r1+ \ !a1+ \ ?r1- \ !a1-) \]  

(A1)

\[ \@(?r2+ \ !a2+ \ ?r2- \ !a2-) \]  

(A2)

\[ \@(!r3+ \ !a3+ \ !r3- \ ?a3-) \]  

(A3)

Note that (A1) and (A2) are two copies of (B); to each of its users the arbiter appears as the user complement. (A3) is a copy of (U); the arbiter reduces two independent users to a single one. This implies that the arbiter has to be deadlock-free (live).

The essence of the arbiter is to exclude a situation where it has granted the requests of both users. In other words, as long as the users follow their protocol, the following