28

Specification and Verification of the PowerScale™ Bus Arbitration Protocol: An Industrial Experiment with LOTOS

Ghassan CHEHAIBAR
Bull S.A., Rue Jean-Jaurès, F-78340 Les Clayes-sous-Bois, France
Current E-mail: Ghassan.Chehaibar@inrialpes.fr

Hubert GARAVEL, *Laurent* MOUNIER
Inria Rhône-Alpes / Verimag, Miniparc-Zirst, rue Lavoisier, F-38330 Montbonnot Saint-Martin, France
E-mail: Hubert.Garavel@imag.fr, Laurent.Mounier@imag.fr

Nadia TAWBI
Bull S.A., Rue Jean-Jaurès, F-78340 Les Clayes-sous-Bois, France
Current E-mail: Tawbi@ift.ulaval.ca

Ferruccio ZULIAN
Bull H.N., I-20010 Pregnana Milanese, Italy
E-mail: F.Zulian@it12.bull.it

Abstract

This paper presents the results of an industrial case-study concerning the use of formal methods for the validation of hardware design. The case-study focuses on POWERSCALE™, a multiprocessor architecture based on POWERPC™ micro-processors and used in Bull's ESCALA™ series of servers and workstations*. The specification language LOTOS (ISO International Standard 8807) was used to describe formally the main components of this architecture (processors, memory controller and bus arbiter). Four correctness properties were identified, which express the essential requirements for a proper functioning of the arbitration algorithm, and formalized in terms of bisimulation relations (modulo abstractions) between finite labelled transition systems. Using the compositional and on-the-fly model-checking techniques implemented in the CADP (CÆSAR/ALDÉBARAN) toolbox, the correctness of the arbitration algorithm was established automatically in a few minutes.

Keywords

FDT-based system and protocol engineering, FDT-application, Practical experience and case studies, Tools and tool support, Verification, validation and testing, LOTOS, Labelled Transition Systems, Model-checking, Bisimulations, Hardware protocols

*PowerScale and Escala are registered trademarks of Bull. PowerPC is a registered trademark of the International Business Machines Corporation.

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

1 INTRODUCTION

The design of hardware systems involves methodologies based upon *hardware description languages* such as VHDL [IEE93] or VERILOG [IEE95]. These languages support various description levels, including architectural, behavioural, register-transfer, gate and switch levels. Many tools exist for simulating the descriptions written in these languages, synthetizing implementations automatically, and generating test sequences.

However, if one is only interested in the high-level functional design of a hardware system, namely the correctness of the distributed algorithms used in hardware systems (e.g., bus arbitration protocols, cache coherence protocols, etc.), then hardware description languages are perhaps not the best candidates for modelling these distributed algorithms. These languages are probably too detailed for this specific problem. Many hardware-related details have to be described, although they are not directly relevant to the algorithms themselves; this may result in overspecification issues. Moreover, since the descriptions are overly complex, the simulation tools handle them often slowly. One may therefore wonder whether the Formal Description Techniques (FDT) defined for computer networks and telecommunication systems could not be also applied with profit to the distributed algorithms used in hardware systems.

In this paper, we investigate the use of LOTOS[†] [ISO88b] for the formal specification and verification of the bus arbitration algorithm used in Bull's POWERSCALE™ architecture. We selected LOTOS for this case-study because its underlying semantics model is based on the rendez-vous paradigm, which is appropriate for the description of hardware entities (processors, memory controllers, etc.) communicating by means of electrical signals sent on wires. The possibility of using LOTOS for the description of hardware systems was already pointed out in [FL93] and [ST93]. For this reason, we used LOTOS rather than other protocol description languages such as ESTELLE [ISO88a] and SDL [IT92] based on infinite FIFO queues, which are clearly not adapted to our problem. There are several possible *specification styles* in LOTOS [VSS88]: for this case-study, we adopted an imperative approach (the so-called *resource-oriented style*) in which LOTOS is used as a concise and readable language to describe a set of extended finite-state machines (i.e., automata with state variables) running in parallel and communicating using rendez-vous. In this approach, each architecture component (processors, bus arbiter, etc.) is represented by an extended finite-state machine, possibly refined into sub-components.

To express the expected functioning properties of the bus arbiter, we identified four correctness requirements, and proved them automatically using the CADP (CÆSAR/ALDÉBARAN) toolbox [FGM+92, FGK+96]. These requirements were expressed using Labelled Transition Systems (LTSs) and the verification process was based on the comparison of LTSs modulo equivalence or preorder relations. This choice was also motivated by the existence in the CADP toolbox of a powerful tool, ALDÉBARAN, for checking bisimulation relations, which was available at the time of our case-study (more recent versions of the CADP toolbox also include evaluators for temporal logic formulas).

This article is organized as follows. Section 2 presents the POWERSCALE™ architecture and gives the main facts about its formal description in LOTOS. Section 3 focuses on the bus arbitration protocol. Section 4 sketches the four correctness properties, informally

[†] *Language Of Temporal Ordering Specification*

first, then formally. After a brief overview of the CADP toolbox, Section 5 presents the verification approach we followed and the results we obtained. Finally, some concludig remarks are drawn in Section 6.

2 THE POWERSCALE ARCHITECTURE

POWERSCALE™ [BR95] is an original, Bull-patented, symmetrical multiprocessor architecture using IBM's POWERPC™ processors. The POWERSCALE™ architecture is used in Bull's ESCALA™ series of servers and workstations. A schematic view of this architecture (reproduced from [BR95] with minor adaptations) is shown on Figure 1.

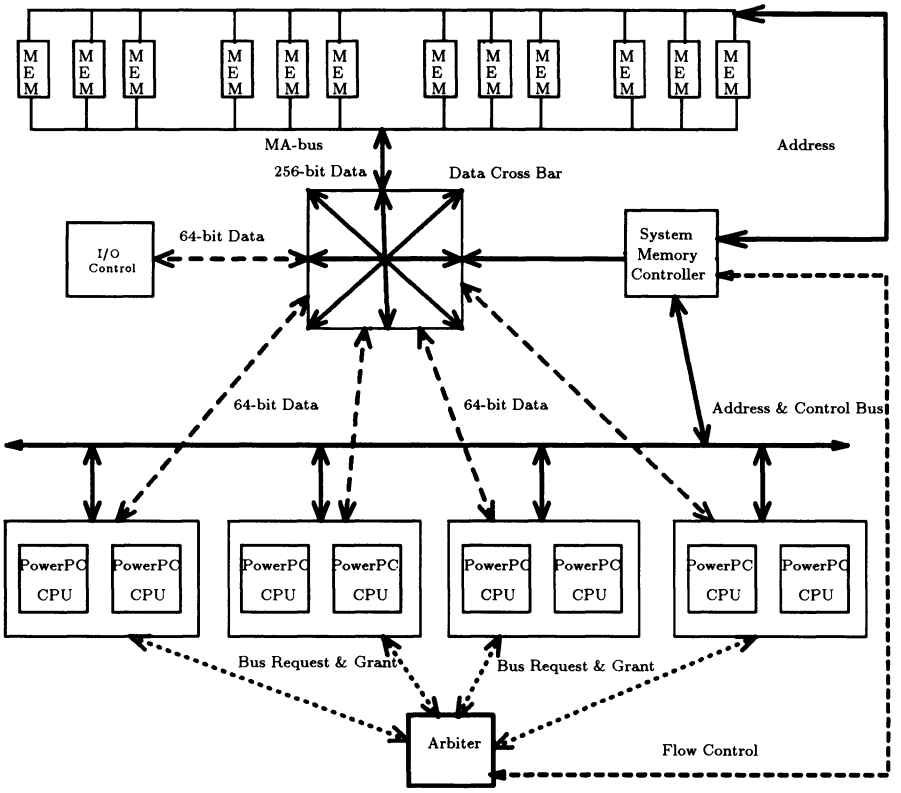

Figure 1 The POWERSCALE™ architecture

The POWERSCALE™ architecture supports up to four *nodes*, each node containing two

POWERPC™ processors. There is also a fifth node specially dedicated to the control of input/output operations.

All the processors share a global memory array. In addition, each processor is equipped with a data cache that allows to speed up accesses to the shared memory. To maintain the consistency between the different caches of the processors, the POWERSCALE™ architecture uses a MESI (*Modified, Exclusive, Shared, Invalid*) snoopy-based protocol (see, e.g., [PP84] with a slightly different terminology) implementing the *weak consistency storage model* [GLL+90].

The processors access the shared memory using two 64-bit buses: an *address bus* and a *data bus*

The address bus is used for *command* transfer. A command is a data structure containing several fields: the *type of operation* (read, write, invalidation of a data stored in a cache, etc.), the *address* concerned by the operation, and various other fields such as: tags associated to the data sent as response to a read operation, bits related to the cache coherence protocol, etc. The precise knowledge of the command format is not required for our case-study.

The data bus is implemented as a crossbar switch (named DCB for *Data Cross Bar*). It centralizes five 64-bit data paths originating from the four nodes and the input/output dedicated node. It is connected to the shared memory via a third 256-bit bus (named MA-bus).

The *System Memory Controller* (SMC) is connected to the address bus and is in charge of controlling the memory, the DCB and the MA-bus. It switches data paths in DCB and orders transfers through the MA-bus.

Finally, the *Arbiter* (ARB) is connected to all the processors and to the SMC. It is in charge of the arbitration of the address bus and data pathes. It takes care to avoid conflicts when granting the address and data buses to the processors and the SMC.

After reading the relevant documents defining the POWERSCALE™ architecture, we produced a 2,000 lines formal description in LOTOS of the overall architecture, together with a 30-pages technical document. Although this description could be compiled and simulated, it was not appropriate for a formal verification, for two reasons: it was too large for being analyzed exhaustively and, at the same time, some critical parts were not detailed enough to allow thorough analysis.

We therefore decided to focus on the bus arbitration protocol, which was a challenging target for both researchers and industrialists because of its complexity and its essential role in the POWERSCALE™ architecture.

3 FORMAL DESCRIPTION OF THE ARBITRATION PROTOCOL

The purpose of this section is to show the different steps that led us to get the abstract model of the POWERSCALE™ arbiter on which we performed our verification.

In order to verify the arbiter functions, we have built a formal description in which the arbiter is detailed, together with its surrounding devices: the processors and the SMC. Therefore, at the highest description level, we consider three types of communicating processes: the arbiter, the processors, and the SMC. However, the behaviour of processors and the SMC are abstracted away to modelize only the interactions with the arbiter.

To keep verification tractable, we made some simplifying assumptions, all of which are compatible with the actual POWERSCALE™ architecture. Firstly, we have only modeled the arbitration mode for POWERPC™ 620 processors, while the actual arbiter has slightly different modes for POWERPC™ 601 and 604 processors.

Secondly, we decided to consider only a single node with two processors (noted P0 and P1 in the sequel). Consequently, the arbiter which we model manages the address bus and a single data path, whereas the actual arbiter manages the address bus and five data paths. This simplification is sound, since the five data paths are managed by the POWERSCALE™ arbiter in a completely independent way. Only the address bus arbitration protocol is concerned by this change, because it must handle two processors instead of ten; however, it was written in a generic way, i.e., it is parameterized by the number of processors.

3.1 The processors

Each processor has access to two buses: the address bus and a data bus. Before accessing each bus, a processor must emit a request and obtain a grant from the arbiter. Depending upon the type of operation to perform, the processor may ask for a single bus or both. Each processor can issue three types of operations:

- In response to an *address bus request* (`ABR`), the processor receives an *address bus grant* (`ABG`): this is the case of address-only operations, which include read operations, cache invalidation requests, etc. For these operations, a command is sent on the address bus.
- In response to a *processor data bus request* (`PDBR`), the processor receives a *processor data bus grant* (`PDBG`): this is the case of *interventions*, i.e. data-only (cache to cache) operations in which a processor P_i provides data to another processor P_j in response to a read operation from P_j.
- In response to an *address-data bus request* (`ADBR`) the processor receives first an `ABG`, then a `PDBG`: this is the case of address-data operations. These are write operations to the memory. The address bus is used to send the command and the data bus is used to send the data. The command and the data are sent asynchronously on the two buses.

In our LOTOS description, we only model the bus requests and grants and forget about the actual operations (read, write, intervention). So, each processor is refined into two communicating sub-processes:

- The process `DATA_SENDER` is in charge of sending data on the data bus. When this process is activated, it can either send a data and free the bus or free the bus without sending data.
- The process `MASTER` is in charge of emitting all bus requests, obtaining bus grants, and sending commands on the address bus. In particular, when obtaining a `PDBG` (for a write operation or an intervention), it activates the `DATA_SENDER` process and can immediately start to emit a new `ABR` (even if the data bus has not been freed yet); therefore, several bus operations can be processed in parallel.

3.2 The System Memory Controller

In the actual POWERSCALE™ architecture, the SMC is connected to the address bus, the data cross bar (DCB), the memory, and the arbiter. But, since we only deal with a single node, we need not model all the details of the DCB; instead, we regard it as a simple bus: hence, we can assume that the SMC is directly connected to the data bus.

For each data path, there are two internal registers in the SMC (called DIRs for *Data In Registers*). A DIR is a one-slot buffer designed to receive data emitted by the processor. When a processor emits a write operation, the corresponding operand (sent by the DATA_SENDER process) is temporarily stored in a DIR until it is written to the memory, which frees the DIR. A processor is not granted a PDBG unless there is a free DIR available in the SMC.

When a processor wants to perform a read operation, the SMC emits a *Memory Data Bus Request* (MDBR) to obtain the data bus connected to this processor and, after obtaining a *Memory Data Bus Grant* (MDBG), orders the data transfer from the memory to the processor.

For the purpose of our case-study, it was not necessary to model the SMC behaviour in full detail. We made appropriate abstractions instead and split the SMC in two communicating sub-processes:

- The process SMC_DI_STAT maintains the number of busy DIRs in the SMC. This number is incremented when a processor sends data. On the opposite, a copy from a DIR to the memory decrements this number. As we have no specific LOTOS process to represent the memory, we consider that the decrementation can occur non-deterministically when the number of busy DIRs is greater or equal to 1.
- The process M_DATA_REQ is in charge of emitting MDBRs and receiving MDBGs. We do not model data sending itself, since it is independent from the arbitration mechanism itself: the important events are requests and grants for the data bus.

3.3 The arbiter

The arbiter is the core of our modelling. When trying to formalize its behaviour, a number of questions arose, which required further technical explanations from the designer of the POWERSCALE™ arbiter.

The arbiter communicates with the processors and the SMC. It manages the address bus and the data bus simultaneously. The address bus is accessed by the processors; the data bus is accessed by the processors and the SMC. The arbitration strategies for both buses are based upon the same *round robin* algorithm: the arbiter maintains a circular list of *devices*[‡] and a *current pointer* in this list; it continuously scans the list, starting from the current pointer and seeking for the first device with a pending bus request. If such a device exists, it is delivered a bus grant and the current pointer is moved to the next device in the circular list. Otherwise, the current pointer is kept unchanged.

In addition to its arbitration role, the arbiter implements a mechanism to control the flow of data sent to the SMC: when both DIRs of the SMC are busy, the arbiter does not

[‡] i.e., the processors and, in the case of the data bus, the SMC

deliver PDBGs to the processors. Hence, the round robin algorithm implemented in the data arbiter is slightly different from the above description. When both of the two DIRs are busy, the arbiter issues grants only for the memory and does not move the current pointer after giving a grant. This mechanism is called *masking*.

In our modelling, we refined the arbiter in four communicating sub-processes:

- The process RND_ADDR_ARB is in charge of the address bus. It receives ABR and ADBR requests from the processors and delivers the corresponding ABG grants to the processors on a round-robin basis. For an ADBR request, when the corresponding ABG is delivered to the processor, an *internal data bus requests* (IDBR) is put in a FIFO queue in order to be served later by the data bus arbiter. This FIFO queue ensures that the PDBG grants generated for ADBR requests are delivered in the same order as ABG grants.
- The process INT_PDBR_FIFO implements a FIFO queue in which the ABG grants generated for ADBR requests are stored, before being transmitted to the data arbiter as IDBR.
- The process RND_DATA_ARB is in charge of the data bus (in the actual POWERSCALE™ architecture, there are five such processes, one per data path). It receives data bus requests originating from the processors (PDBR), from the SMC (MDBR), or from the FIFO queue (IDBR). It delivers grants on a round-robin basis and taking into account the number of busy DIRs in the SMC (if the two DIRs are busy, the current pointer is not moved after an MDBG).
- The process ARB_DI_STAT maintains the number of busy DIRs according to signals it gets from SMC_DI_STAT. It informs the data arbiter whether to deliver PDBGs to the processors or not.

3.4 The LOTOS description

A LOTOS description representing the arbitration protocol (arbiter, processors and SMC) was developed. This specification contains 760 lines of code (including a few comment lines), divided into 200 lines (26%) for the data part and 560 lines (74%) for the control part. The data part contains 6 type definitions (3 enumerated types, 2 tables used for the round-robin algorithm, and 1 FIFO queue) and 14 process definitions (corresponding to the aforementioned processes and their parallel combinations at different levels).

Figure 2 gives an overview of the arbitration protocol. Only a single, generic processor is represented; it is noted P !*pid*, where *pid* is a parameter denoting the index of the processor (0 or 1). The arbiter, the generic processor, and the SMC are refined in sub-processes, as explained above. Boxes represent LOTOS processes and lines between boxes represent communication gates between processes. For instance, the processor P !*pid* and the arbiter communicate via the gate labelled ABR !*pid*, which expresses the fact that each processor can send to the arbiter an address bus request ABR parametrized by its index.

4 BEHAVIOURAL EXPRESSION OF THE REQUIREMENTS

Before performing verification, it is necessary to define which functioning properties have to be verified. We identified four requirements related to bus arbitration and data transfer from the different devices (processors and memory). We found more suitable and easier to

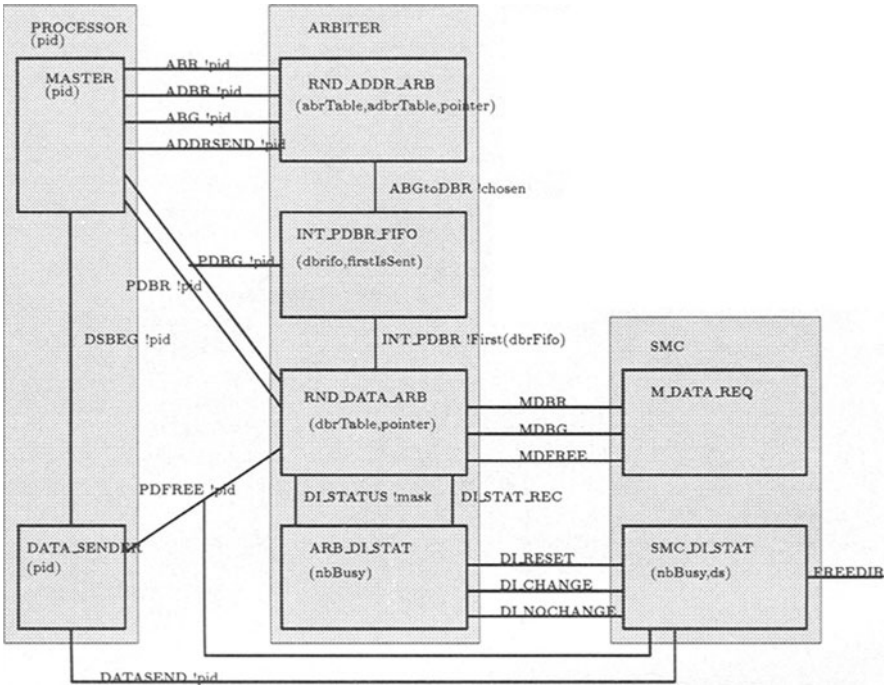

Figure 2 The LOTOS program structure

describe these requirements using behavioural specifications rather than temporal logic formulas. We adopted the following approach: from an abstract point of view, let us assume that we can translate the whole LOTOS description of the arbitration protocol into a (large) LTS, which we will note `arb` in the sequel. Then, for each requirement, we define another, much smaller LTS, which we will note `req`. To express that the arbitration protocol satisfies this requirement, we state that `arb` and `req` are related modulo a given equivalence relation or a given preorder relation.

Notice that, as the various LTSs `req` are generally small, they can be specified simply by listing their states and transitions. In our case-study, we chose to specify them directly in LOTOS and to use software tools to automatically generate the corresponding LTSs. This allowed us to write generic requirements, parameterized with a processor identifier *pid*, which can be easily instantiated with either P0 or P1. However, in this paper, we show the LTSs themselves, rather than the LOTOS code from which they were produced.

When comparing `arb` and `req`, abstraction criteria have to be used, as `arb` contains many details which are not relevant to the requirement being expressed. Thus, LTSs have

to be abstracted when compared, by *hiding* all transitions that are not to be observed. For instance, if we want to prove properties about the address arbitration fairness, we need to hide all transitions but **ABR** and **ABG**. In the sequel, we adopt the following convention: any action in **arb** that is not present in **req** is to be hidden. Hence we observe only actions related to the property we want to prove.

We now present the four requirements and their expression in terms of equivalence or preorder relations:

Proper response to bus requests: When a processor Pi issues a bus request, it is always possible to satisfy this request by delivering to Pi the corresponding grant(s). More precisely, each **ABR** !Pi may be followed by an **ABG** !Pi; each **PDBR** !Pi may be followed by a **PDBG** !Pi; each **ADBG** !Pi can be followed by an **ABG** !Pi and then a **PDBG** !Pi. These response properties state that it is *always possible* for each bus request to be followed by the corresponding bus grant(s); this proves the *deadlock-freeness* of the arbiter in addition to proper bus granting. These properties do not state that bus grant(s) are *eventually* delivered, due to the presence of τ-cycles in the model caused by the non-deterministic interleaving semantics and absence of fairness assumption.
To express this requirement for processor P0, we state that the abstracted **arb** should be *branching equivalent* [vGW89] to the graph **req** shown on Figure 3. We use branching equivalence because it preserves the deadlocks.
We express the same property for processor P1 by using another **req** in which P0 and P1 are interchanged.

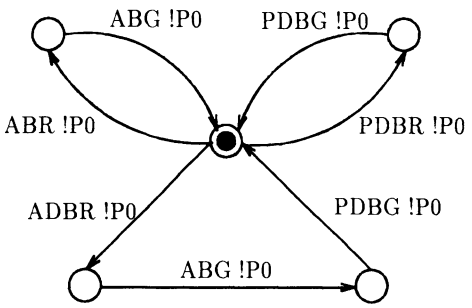

Figure 3 Property graph #1 (Proper response to bus requests)

Fairness of the arbitration: While a processor Pi has issued a bus request and is waiting for the corresponding grant(s), at most one bus request emitted by the other processor can be granted. For instance, if an **ABR** (resp. **PDBR**) request of P0 is waiting, at most one **ABG** (resp. **PDBG**) grant may be delivered to P1 before an **ABG** (resp. **PDBG**) grant is delivered to P0.
To express this requirement for the address bus and processor P0 only, we state that the abstracted **arb** should be included, modulo the safety preorder [BFG+91], in the

graph **req** shown on Figure 4. We also need to verify three similar properties, for the data bus as well, and by interchanging processors P0 and P1.

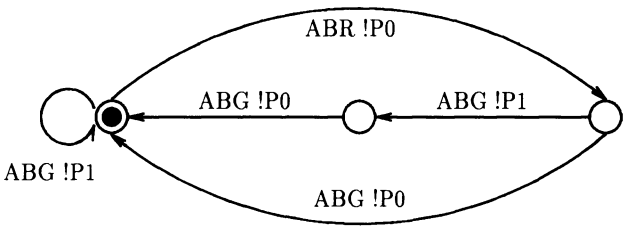

Figure 4 Property graph #2 (Fairness of the arbitration)

Order of grants for address-data requests: When both processors P0 and P1 issue ADBR requests, the PDBGs should be delivered in the same order than the ABGs. For instance, if ABG !P0 precedes ABG !P1, then PDBG !P0 should also precede PDBG !P1.
To express this requirement for processor P0, we state that the abstracted **arb** should be included, modulo the safety preorder, in the graph **req** shown on Figure 5. This guarantees that all execution trees of the arbiter are covered by **req**. The same property should hold when P0 and P1 are interchanged.

Correctness of the DBG flow control: When both processors are granted the data bus, they can send data that will be stored in the two DIRs of the SMC (see Section 3.2 above). The correctness of the flow control mechanism is expressed by two properties: (a) it is not possible to send a data when the two DIRs are busy (which implies that no PDBG is delivered when the two DIRs are busy); (b) it is always possible to free a DIR so that data sending becomes possible.
To express these two properties at once, we state that the abstracted **arb** should be branching equivalent to the graph **req** shown on Figure 6, which is nothing but a two-slot buffer.

Instead of branching equivalence, other equivalences, such as the well-known observation equivalence [Mil89]) could have been used. We prefered branching equivalence because there exist efficient algorithms for it [GV90, Mou92], some of which are implemented in ALDÉBARAN. Although branching equivalence is stronger than observational equivalence in the general case, both equivalences coincide if the property graph does not contain τ-transitions [Mou92], which is the case here.

Similarly, trace inclusion could have been used instead of safety preorder. We prefered the latter since it is efficiently implemented in ALDÉBARAN. Moreover, both relations coincide when the property graphs are deterministic, which is the case here.

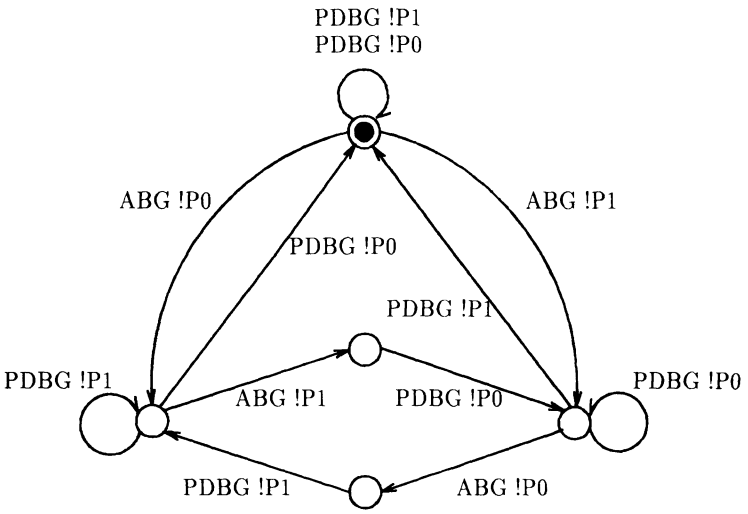

Figure 5 Property graph #3 (Order of grants for address-data requests)

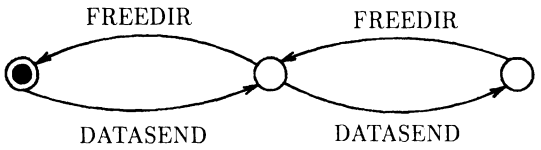

Figure 6 Property graph #4 (Correctness of the DBG flow control)

5 VERIFICATION

In this section, we briefly present the approach used to verify the LOTOS description of the arbitration protocol. For this case-study, we used only a subset of CADP toolbox, namely the CÆSAR.ADT [Gar89, GT93], CÆSAR [GS90], XSIMULATOR, and ALDÉBARAN [Fer90, FKM93] tools. CÆSAR.ADT and CÆSAR are LOTOS-to-C compilers; additionally, CÆSAR can generate an LTS corresponding to a LOTOS description. XSIMULATOR is an interactive, X-WINDOWS-based simulator, offering unlimited backtracking facilities. ALDÉBARAN compares two LTSs with respect to equivalence or preorder relations; an important feature of ALDÉBARAN is on-the-fly verification: it accepts as input a system defined by a composition of LTS and can compare this system to another one without building the corresponding LTS.

Once the LOTOS description was written, we performed a first debugging by compiling its data part (using the CÆSAR.ADT compiler), compiling its control part (using the CÆSAR compiler), and analyzing a subset of its behaviour (using XSIMULATOR). XSIMULATOR revealed some deadlocks, which have been fixed.

As regards performances, all our experiments were carried out on a low-end Sparc machine, with 40 Mbytes of main memory. From the LOTOS description (720 lines, 32 kbytes), CÆSAR.ADT generated a C file for the data types (1,044 lines, 42 kbytes) and CÆSAR generated a C file for the behaviour part (2,241 lines, 92 kbytes). Linking and compiling these C files together produced a small executable program (49 kbytes). Performing the whole translation and starting the simulation takes less than one minute.

However, interactive simulation is not sufficient to ensure the correctness, as it only gives a very limited coverage of all possible execution sequences. To perform full verification, we tried to generate exhaustively `arb` (the LTS of the arbitration protocol) using CÆSAR. This "brute-force" approach failed due to memory limitations, after generating 580,000 states and 1,540,000 transitions approximately.

We therefore switched to another *compositional, on-the-fly* approach, based on a *divide and conquer* paradigm. We split the arbitration protocol into three parallel components noted `ARB_COMP_1`, `ARB_COMP_2`, and `ARB_COMP_3`. This decomposition is shown on Figure 7. It is worth noticing that the splitting is not done according to the hardware components (the arbiter, the processors and the SMC), but in a transversal way, by grouping together sub-processes belonging to different hardware components. There are many possible decompositions; we present here the one we found to give satisfactory results. A rule of thumb is to put together the processes which constraint each other, i.e., which have many interactions together: this reduces the size of the generated LTS.

For each of the three parallel components we generated the corresponding LTS using CÆSAR and reduced this LTS modulo strong bisimulation using ALDÉBARAN. This happened to be tractable since the complexity of each component remains within the amount of memory available on our machine. The following table gives, for each component, the number of states S and the number of transitions T of the LTS generated by CÆSAR, the number of states S' and the number of transitions T' of the LTS reduced by ALDÉBARAN, as well as the durations G and R spent for generating and reducing these LTSs.

component	S	T	S'	T'	G	R
`ARB_COMP_1`	176,810	566,270	6,746	21,191	8 mn	25 mn
`ARB_COMP_2`	8,882	32,768	183	427	< 1 mn	< 1 mn
`ARB_COMP_3`	588	1,798	237	687	< 1 mn	< 1 mn

Thanks to this decomposition, the problem is reduced to the verification of a system consisting of three communicating state machines. We tried to generate directly the entire LTS (noted `red_arb`) for this system, but this failed again: the number of states of `red_arb` is potentially high ($6,746 \times 183 \times 237 \approx 2.9\ 10^8$ states).

We therefore used ALDÉBARAN to compare on-the-fly this system of three communicating processes with each `req` graph expressing the expected properties. On-the-fly

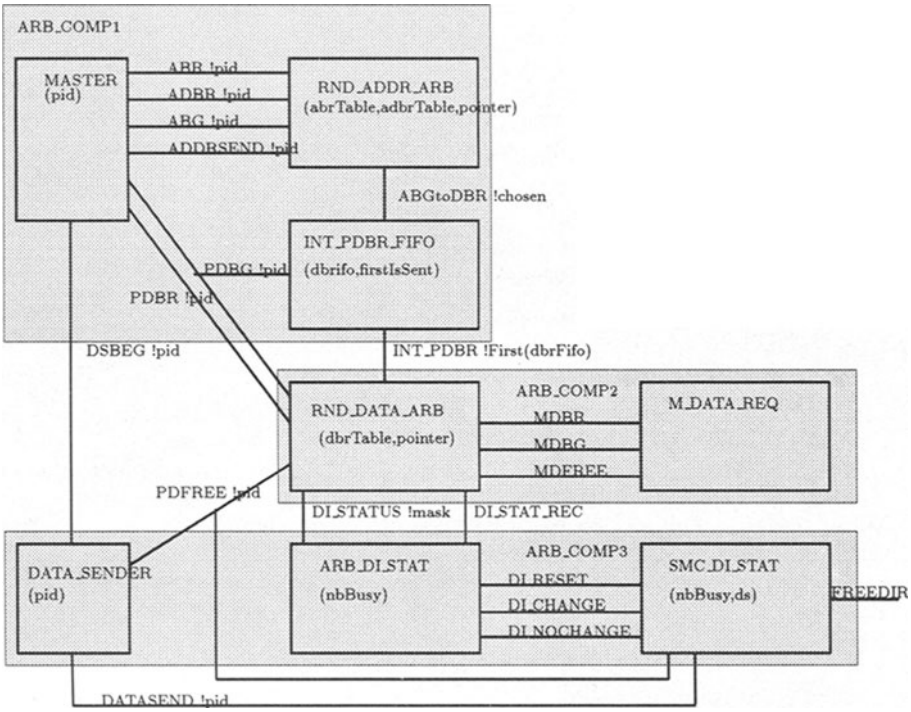

Figure 7 The LOTOS program decomposition

comparison means that ALDÉBARAN does not generate `red_arb` entirely: it only explores those parts of `red_arb` that are relevant to the property expressed by `req`.

Using ALDÉBARAN, we proved that all the requirements stated in Section 4 were satisfied. Each property was verified in less than one minute.

Then, we modified our LOTOS description to implement a different version of the data arbitration algorithm, in which the current pointer is always moved to the next item in the circular list after delivering an `MDBG`, which means that this different version does not implement the masking mechanism described in Section 3.3. This version of the algorithm was considered one moment by the designers of the POWERSCALE™ architecture, but it was discarded and not implemented. ALDÉBARAN discovered that, for this modified algorithm, the fairness property was no longer satisfied.

6 CONCLUSION

This paper reports the results of an industrial experimentation of formal methods. The aim of this case-study was to investigate whether the Formal Description Technique LOTOS and the protocol-engineering toolbox CADP were mature enough for being applied to real, industrial applications, such as the multiprocessor systems developed by Bull.

In a first time, we have described formally, using LOTOS the POWERSCALE™ multiprocessor architecture used in Bull's ESCALA™ series. Then, we focused in more detail on the POWERSCALE™ bus arbitration protocol, using appropriate abstractions to cut down the complexity of the problem.

We identified four correctness requirements for the arbitration functionality, which we expressed in terms of equivalence and preorder relations between labelled transition systems.

Verification was performed automatically using the CÆSAR and ALDÉBARAN tools. For each requirement, expressed in LOTOS, we generated the corresponding LTS. Due to lack of memory space, we have not been able to do the same for the arbitration protocol, since its LTS was much too large for being generated. We used instead a compositional verification approach, by splitting the LOTOS description into three parts, the LTSs of which could be generated and minimized separately. Then, these LTSs were combined together and compared on-the-fly against the requirements. By doing so, we were able to prove the correctness of the arbitration protocol. This protocol was already tested and simulated, which explains that no misconceptions were found. However, we discovered an error in a proposed variant of the bus arbiter (which is not actually implemented in Bull products).

This case study was performed in a relatively short lapse of time. Producing the first LOTOS description (whole POWERSCALE™ architecture) took 8 man.months, including the time spent in learning both POWERSCALE™and LOTOS. Producing the second LOTOS description (arbitration functionality) took 1.5 man.months only, including the preliminary debugging using interactive simulation. Requirement capture and verification took about 1.5 man.months. The case-study was facilitated by the complementary expertises brought by the different authors: F. Zulian designed the POWERSCALE™ bus arbiter, G. Chehaibar and N. Tawbi performed the modelling and verification, H. Garavel and L. Mounier provided insights in using the CADP tools and expressing the requirements.

The results of this experiment are encouraging. It seems that LOTOS is appropriate for the description of hardware protocols and that the compositional and on-the-fly verification techniques implemented in the CADP tools allow to deal with mid-size industrial cases involving a fair degree of parallelism.

In this experiment, formal description and verification took place after the arbiter was already designed. In the near future, we intend to apply this approach to a cache coherency protocol for a new Bull architecture under development. We take aim at a complete technology transfer, by progressively integrating formal methods in the existing development process.

ACKNOWLEDGEMENTS

This work has been done in the framework of DYADE, the Bull-Inria Advanced Research Joint Venture. It has been supported by the Bull R&D POWERPC™ Technology Platforms Division, headed by Angelo Ramolini. The development of the CADP tools has been supported in part by the European Commission, under project ISC-CAN-65 "EUCALYPTUS-2: *A European/Canadian* LOTOS *Protocol Tool Set*".

REFERENCES

[BFG+91] Ahmed Bouajjani, Jean-Claude Fernandez, Susanne Graf, Carlos Rodríguez, and Joseph Sifakis. Safety for Branching Time Semantics. In *Proceedings of 18th ICALP*, Berlin, July 1991. Springer Verlag.

[BR95] P. Bennett and A. Ramolini. The PowerScale Architecture: A Technical Overview. *Journal of Technical Information for the Distributed Computing Model*, January-February 1995.

[Fer90] Jean-Claude Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13(2–3):219–236, May 1990.

[FGK+96] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, August 1996.

[FGM+92] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A Toolbox for the Verification of LOTOS Programs. In Lori A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering ICSE'14 (Melbourne, Australia)*, pages 246–259, New-York, May 1992. ACM.

[FKM93] Jean-Claude Fernandez, Alain Kerbrat, and Laurent Mounier. Symbolic Equivalence Checking. In C. Courcoubetis, editor, *Proceedings of the 5th Workshop on Computer-Aided Verification (Heraklion, Greece)*, volume 697 of *Lecture Notes in Computer Science*, Berlin, June 1993. Springer Verlag.

[FL93] M. Faci and L. Logrippo. Specifying Hardware in LOTOS. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of the the 11th International Conference on Computer Hardware Description Languages and their Applications (Ottawa, Ontario, Canada)*, pages 305–312, Amsterdam, April 1993. North-Holland.

[Gar89] Hubert Garavel. Compilation of LOTOS Abstract Data Types. In Son T. Vuong, editor, *Proceedings of the 2nd International Conference on Formal Description Techniques FORTE'89 (Vancouver B.C., Canada)*, pages 147–162, Amsterdam, December 1989. North-Holland.

[GLL+90] K. Gharachorloo, D. Lenosky, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, 1990.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th*

International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada), pages 379–394, Amsterdam, June 1990. IFIP, North-Holland.

[GT93] Hubert Garavel and Philippe Turlier. CÆSAR.ADT : un compilateur pour les types abstraits algébriques du langage LOTOS. In Rachida Dssouli and Gregor v. Bochmann, editors, *Actes du Colloque Francophone pour l'Ingénierie des Protocoles CFIP'93 (Montréal, Canada)*, 1993.

[GV90] Jan Friso Groote and Frits Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In M. S. Patterson, editor, *Proceedings of the 17th ICALP (Warwick)*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638, Berlin, 1990. Springer Verlag.

[IEE93] IEEE. Standard VHDL Language Reference Manual. IEEE Standard 1076-1993, Institution of Electrical and Electronic Engineers, 1993.

[IEE95] IEEE. Verilog HDL Language Reference Manual. IEEE Draft Standard 1364, Institution of Electrical and Electronic Engineers, October 1995.

[ISO88a] ISO/IEC. ESTELLE — A Formal Description Technique Based on an Extended State Transition Model. International Standard 9074, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[ISO88b] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[IT92] ITU-T. Specification and Description Language (SDL). ITU-T Recommendation Z.100, International Telecommunication Union, Genève, 1992.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[Mou92] Laurent Mounier. *Méthodes de vérification de spécifications comportementales : étude et mise en œuvre.* Thèse de Doctorat, Université Joseph Fourier (Grenoble), January 1992.

[PP84] M. S. Papamarcos and J. H. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proceedings of the 11th International Symposium on Computer Architecture*, 1984.

[ST93] Richard O. Sinnott and Kenneth J. Turner. DILL: Specifying Digital Logic in LOTOS. In Richard L. Tenney, Paul D. Amer, and M. Umit Uyar, editors, *Proceedings of the 6th International Conference on Formal Description Techniques FORTE'93 (Boston, MA, USA)*, pages 71–86, Amsterdam, October 1993. North-Holland.

[vGW89] R. J. van Glabbeek and W. P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics (extended abstract). CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989. Also in proc. IFIP 11th World Computer Congress, San Francisco, 1989.

[VSS88] C. Vissers, G. Scollo, and M. van Sinderen. Architecture and Specification Style in Formal Descriptions of Distributed Systems. In S. Aggarwal and K. Sabnani, editors, *Proceedings of the 8th International Workshop on Protocol Specification, Testing and Verification (Atlantic City, NJ, USA)*, pages 189–204, Amsterdam, 1988. IFIP, North-Holland.