Identification of multi-core interference

Frédéric Boniol, Claire Pagetti, Nathanaël Sensfelder

Identification of multi-core interference

Frédéric Boniol Claire Pagetti Nathanaël Sensfelder

November 22, 2018

Abstract

The CAST-32A provides some guidelines to help certify multi-core-based systems in the avionics domain. One major requirement is to compute all the potential interference and to provide adequate mitigation means. In this paper, we compare two approaches to identify the interference: the *initiator-target* and the Phylog models. The latter is more compact and efficient, despite also covering all of the problematic conflictual situations.

Keywords: Multi-core, certification, timing interference

1 Introduction

The last decade has seen the emergence of multi-core processors, i.e. chips integrating several cores linked by a shared interconnect. Although these architectures have been shown to provide huge gains in performance, they have severe lapses in time predictability [20, 21], one of the key elements of certification expectations.

1.1 Identification of potential interference

Aeronautic certification authorities, in association with industrial manufacturers, have published the Multi-Core Certification Review Item (MCP-CRI) [11] (also published as the CAST-32A position paper [7]), in order to provide a set of guidances for software planning and verification on multi-core chips.

Due to resource sharing, couplings exist at the platform level. These can cause *interference* between applications, which, in turn, may lead to unexpected delays, and even the alteration or loss of data. These three issues are not acceptable in the aeronautics domain and must thus be avoided. In terms of certification, this entails a four steps process: First, the applicant must identify all *interference channels*. In the CAST-32A terminology, an interference channel is *a platform property that may cause interference between independent applications*. Second, the applicant must classify the interference as either *acceptable*, *tolerable*, or *unacceptable*. Third, for each *unacceptable* interference, they must provide a mean of mitigation to prevent the system from having catastrophic behaviors. In that context, mitigation signifies that some mechanisms have been proposed to forbid *unacceptable* interference or reduce their effect to *acceptable* or *tolerable* levels. For example, if a resource being accessed in parallel by more than two requesters would lead to a non-acceptable delay, mitigation could take the form of a run-time mechanism that sequentializes the access. Fourth and final, the applicant must argue why the means of mitigation are adequate and why *unacceptable* interference will never occur during aircraft operations.

This requirement is called *resource usage 3* in the CAST-32A. In the sequel, we will only focus on this particular objective and, more precisely, on the identification of interference.

1.2 Objectives and contribution

To the best of our knowledge, few works have proposed solutions for *resource usage 3*. Researchers from Thales have proposed the *Initiator-Target Model* [6, 14, 17, 18] to help identify the interference channels on multi-core chips. Their model is very simple, but suffers from a combinatorial explosion.

Phylog is a French project (2016-2020), funded by the French civil aeronautic agency (DGAC), which aims at offering a model-based and software-aided certification framework for aeronautics systems based on multi/many-core architectures. In [3], we have defined the premises of the Phylog model, presented the notions of interference channels and transactions, and shown an automated process to find the interference channels through the use of Weird [4].

The objective of this paper is to compare and link the *initiator-target* and PHYLOG models. For that purpose, we start with a formal definition of the *initiator-target model* (see section 2). We then refine and formalize our former definition of interference channels (see section 3). We then show that our representation is more compact than the *initiator-target model* despite remaining as expressive. Indeed, our interference channels are the representative elements of the equivalence classes of an equivalence relation (see section 4). All our formalization and computation are supported by implementations made in IDP [10] (see section 5), used as a replacement for WEIRD [4].

2 The initiator-target model

The initiator-target model has been introduced in [6] and reused in [14, 17], and [18]. The goal was to provide a theoretical view for the identification of the interference channels (called *performance contentions* in [6]) that can occur in a multi-core processor.

2.1 Overview

According to their definition, a multi-core is composed of three types of components:

- *Smart initiator* components, i.e. components which can initiate single transactions through the architecture to *target* components. Processing cores (CPU) are examples of smart initiator components. They can, for instance, initiate memory access transactions to memory controllers.
- *Non smart initiator* components, i.e. initiator components which can only initiate dual transactions (i.e., with two targets at the same time). DMA are examples of non smart initiator components.
- *Target* components, i.e. end-components targeted by smart or non smart initiators.

Intermediate components (such as internal buses) between initiators and targets are simply ignored.

Example 1 *Let us illustrate the initiator-target model on the simple architecture shown in Figure 1, composed of two smart initiators (the two CPUs), two non smart initiators (the two DMAs), and four targets (the two memory controllers, the PCIe controller, and one L3 cache used as SRAM memory).*

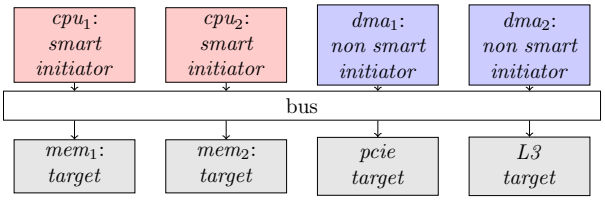


Figure 1: Architecture P_1

Notations 1 *For the sake of readability, our figures use the following color code: smart initiators are in red, non smart initiators are in blue, targets are in gray, and other components are in white.*

An interference channel is seen as a combination of *single test classes* (or *single test cases*) where an interference occurs. A single test class is a transaction initiated by an initiator and targeting a target (for smart initiators) or two targets (in case of non smart initiators).

Example 2 *In the architecture depicted Figure 1,* $cpu_1 \rightsquigarrow mem_1$, $cpu_2 \rightsquigarrow mem_2$, *and* $pcie \rightsquigarrow dma_1 \rightsquigarrow mem_2$ *are examples of single test classes. The first two may denote either read or write transactions from* cpu_k *to* mem_k. *The last one may denote a data transfer from the PCIe device to the second memory through the* dma_1.

$$(cpu_1 \rightsquigarrow mem_1) \parallel (cpu_2 \rightsquigarrow mem_2) \parallel (pcie \rightsquigarrow dma_1 \rightsquigarrow mem_2)$$

is a test class composed of three single test classes running in parallel.

2.2 Formalization

Let us now introduce a set-based formalization of the initiator-target model.

Definition 1 (Initiator-target model) *In the* initiator-target *model, an architecture P is defined by $P = (\mathcal{C}, \rightarrow)$ where*

- *$\mathcal{C} = SI \cup NSI \cup T \cup O$ with SI being the set of smart initiators, NSI the set of non smart initiators, T the set of targets, and O the set of other components. All those sets are disjoint. In the sequel, we will note $n_{SI} = card(SI)$, $n_{NSI} = card(NSI)$ and $n_T = card(T)$;*
- *$\rightarrow \subseteq \mathcal{C} \times \mathcal{C}$ are the hardware connections between components.*

Definition 2 (Single test class) *For an architecture $P = (\mathcal{C}, \rightarrow)$, a single test class for a smart initiator is a pair $(i, t) \in SI \times T$ such that there exists a path in P from i to t, i.e. $i \rightarrow^* t$. In the sequel, we write indifferently (i, t) or $i \rightsquigarrow t$.*

A single test class for a non smart initiator is a triplet $(i, t_1, t_2) \in NSI \times T \times T$ such that there exist a path in P from i to t_2 and one to t_1, i.e. $i \rightarrow^* t_2$ and $i \rightarrow^* t_1$. In the sequel, we write indifferently (i, t_1, t_2) or $t_1 \rightsquigarrow i \rightsquigarrow t_2$.

Definition 3 (Test classes) *Let $P = (\mathcal{C}, \rightarrow)$ be an architecture. A test class is a set of n single test classes and disjoint initiators. For instance, a test class t_c of size 2 is of form $t_c = \{(i_1, t_1), (i_2, t_2)\}$ or $t_c = \{(i_1, t_1), (i_2, t_2, t_3)\}$ or $t_c = \{(t_1, i_1, t_2), (i_2, t_3, t_4)\}$ with $i_1 \neq i_2$. In the sequel, we write indifferently $\{(i_1, t_1), (i_2, t_2)\}$ or $(i_1 \rightsquigarrow t_1) \parallel (i_2 \rightsquigarrow t_2)$.*

An interference channel *is a test class composed of 2 or more single test classes.*

Definition 4 *Let $P = (\mathcal{C}, \rightarrow)$ be an architecture. Let us note TC_∞^P the set of test classes and TC_n^P those of size n:*

$$TC_\infty^P = \bigcup_{n=1}^{n_{SI}+n_{NSI}} TC_n^P$$

Proposition 1 (Total number of test classes [6]) *The number of all possible test classes of P is:*

$$card(TC_\infty^P) = (1 + n_T)^{n_{SI}} \cdot (1 + n_T^2)^{n_{NSI}} - 1 \quad (1)$$

Example 3 *Let us once again consider the architecture P_1 shown in Figure 1: $n_{SI} = 2$, $n_{NSI} = 2$, $n_T = 4$, $\rightarrow$= {(cpu$_1$,bus), (cpu$_2$,bus), (dma$_1$, bus), (dma$_2$,bus), (bus,mem$_1$), (bus,mem$_2$), (bus,L3), (bus,pcie)}. Applying equation 1 yields:*

$$card(TC_\infty^{P_1}) = (1 + 4)^2 \cdot (1 + 4^2)^2 - 1 = 7224$$

meaning that the interference analysis may need up to 7224 test classes to be analyzed on this architecture. Aeronautic certification standards require the assessment of the worst case execution time (WCET) for the critical software functions running on the cores of the processor. As stated in the introduction, interference may strongly affect this execution time. It is up to the designer to characterize the severity *of each interference with respect to the execution time of each software function, and, at the end, to show that all the* unacceptable *interference (i.e., the ones that induce too high WCETs) are properly mitigated by appropriate means (e.g., arbiters, time-triggered execution schemes, etc.). In the case of the (rather small) architecture P_1, such an assessment requires the investigation of the 7224 test classes.*

2.3 Hypotheses

In the seminal paper, there was no specific rule about the reachability of a target by an initiator. Implicitly, the authors assumed that all targets were reachable by all initiators. Moreover, they did not make a distinction on the type of transactions (e.g. read or write). Finally, they assumed a unique path from any given initiator to any given target, which is not the case in many-cores. Most commercial multi-core processors satisfy these two hypotheses. Let us formalize them:

- **Hyp1.** All targets are reachable by all initiators: $\forall i \in SI \cup NSI, \forall t \in T, i \rightarrow^* t$
- **Hyp2.** There is a unique path from an initiator to any non initiator component: $\forall i \in SI \cup NSI, \forall t \in O \cup T$, if $i \rightarrow a_1 \rightarrow \ldots \rightarrow a_n \rightarrow t$ and $i \rightarrow b_1 \rightarrow \ldots \rightarrow b_m \rightarrow t$, then $n = m$ and $\forall k$, $a_k = b_k$.

Proposition 2 *The reachability relation $\rightarrow$ satisfying **Hyp2** is acyclic and defines a partial order $<_P$: $\forall \alpha, \beta \in SI \cup NSI \cup T \cup O, \alpha <_P \beta \Leftrightarrow \alpha \rightarrow^* \beta$.*

3 PHYLOG model

In the PHYLOG project, we need to tackle the identification of all interference channels as in Brindejonc et al.'s approach.

3.1 Overview

We believe that the current *initiator-target model* is insufficient as is and that it must be enriched.

1. Brindejonc's approach is a black-box approach: it does not consider internal components. Two architectures with different topologies may be characterized by the same test classes, even when their interference differ.

2. Test classes do not necessarily lead to any actual interference between transactions. For instance, in the architecture P_2 depicted Figure 2, the test class $(cpu_1 \rightsquigarrow mem_1) \parallel (pcie \rightsquigarrow dma_1 \rightsquigarrow L3)$ does not cause any contention as these transactions cross two different buses in parallel without interfering with each other. Many other test classes in P_2 are *interference-free* as well.

3. Their approach is simple, but suffers from scalability issues: For a T4240 processor, composed of 12 cores ($n_{SI} = 12$), 3 DMAs ($n_{NSI} = 3$), 2 memory controllers, 1 PCIe, and 1 L3 cache used as SRAM memory ($n_T = 4$), there are more than 10^{12} test cases.

Ideally, the number of test classes to be explored should be as low as possible. We thus propose grouping them according to the interference they cause on the components.

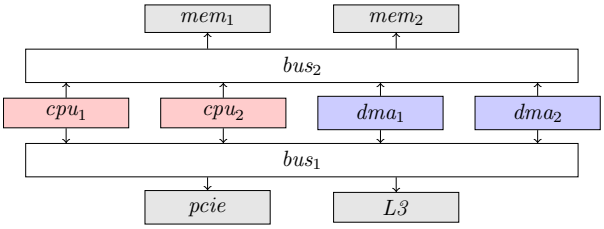


Figure 2: Architecture P_2

3.2 Transaction model

Definition 5 (Transaction) *For an architecture $P = (\mathcal{C}, \rightarrow)$, a transaction is a finite branching word of components*

$$tr = i \cdot ((b_1 \dots b_n) + (c_1 \dots c_m))$$

with $i \in SI \cup NSI$; for all j, $b_j \in O \cup T$, $(b_j, b_{j+1}) \in \rightarrow$ and $(i, b_1) \in \rightarrow$. If $i \in SI$, $c_1 \dots c_m = \epsilon$ is the empty word; otherwise, for all j, $c_j \in O \cup T$, $(c_j, c_{j+1}) \in \rightarrow$ and $(i, c_1) \in \rightarrow$.

In the following, we will consider that the branching operator "+" is commutative. That is, $tr = i \cdot ((b_1 \dots b_n) + (c_1 \dots c_m)) = i \cdot ((c_1 \dots c_m) + (b_1 \dots b_n))$.

Example 4 *A single test class is a transaction from an initiator to one or two targets. In P_1, the single test class* $\text{cpu}_1 \rightsquigarrow \text{mem}_1$ *is the transaction* $\text{cpu}_1.(\text{bus.mem}_1 + \epsilon)$*; whereas* $\text{pcie} \rightsquigarrow \text{dma} \rightsquigarrow \text{mem}_2$ *is the transaction* $\text{dma}.(\text{bus.mem}_2 + \text{bus.pcie})$.

Definition 6 *We define a series of useful functions for a transaction $tr = i \cdot ((a_1 \dots a_n \cdot b_1 \dots b_m) + (a_1 \dots a_n \cdot c_1 \dots c_p))$ with for all l, k $b_l \neq c_k$:*

- *$hd(tr) = i$ (head of the transaction),*
- *$prefix(tr) = a_1 \dots a_n$ (common prefix of the transaction, possibly empty).*
- *$proj_1(tr) = i \cdot a_1 \dots a_n \cdot b_1 \dots b_m$ (first branch of the transaction),*
- *$proj_2(tr) = i \cdot a_1 \dots a_n \cdot c_1 \dots c_p$ (second branch),*

- $lg(tr) = \{a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_m\}$ *(language of the transaction, i.e., the set of components involved),*
- $tl(\alpha_1 \ldots \alpha_n) = \alpha_n$ *(tail of a word),*
- $pred(\alpha_1 \ldots \alpha_{n-1} \cdot \alpha_n) = \alpha_{n-1}$ *(penultimate component of a word).*

According to hypothesis **Hyp2**, each tc is modeled by a unique transaction tr in the PHYLOG model.

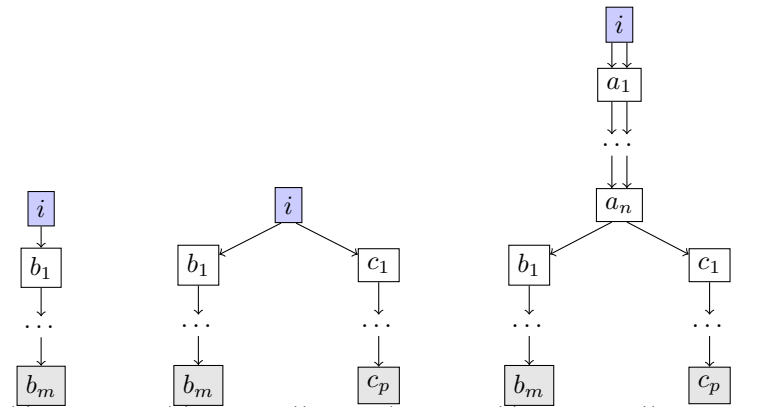
Definition 7 (Transactions associated to a single test class) *When $(i, t) \in SI \times T$, the associated* PHYLOG *transaction is defined as $phy(i, t) = tr$ with $hd(tr) = i$, $tl(\text{proj}_1(tr)) = t$ and $\text{proj}_2(tr)) = \epsilon$.*

When $(i, t_1, t_2) \in NSI \times T \times T$, the associated PHYLOG *transaction is defined as $phy(i, t_1, t_2) = tr$ with $hd(tr) = i$, $tl(\text{proj}_1(tr)) = t_1$ and $tl(\text{proj}_2(tr)) = t_2$.*

Proposition 3 *Let $P = (\mathcal{C}, \rightarrow)$ be an architecture satisfying **Hyp1** and **Hyp2**. Let $tr = i \cdot ((b_1 \ldots b_n) + (c_1 \ldots c_m))$ be a transaction in P. Then:*

1. *$\forall j \neq k, b_j \neq b_k$ (a component can appear at most once in the b branch),*
2. *either $(c_1 \ldots c_m) = \epsilon$ (in the case of a smart initiator) or $\forall j \neq k, c_j \neq c_k$ (same as with the b branch),*
3. *if there exist j, k such that $b_j = c_k$, then $j = k$ and $\forall l < k, b_l = c_l$ (either the two branches do not share any component, or they share a common prefix).*
4. *$\forall l, l' > max\{j \mid b_j = c_j\}, b_l \neq c_{l'}$ (after the common prefix, if it exists, the two branches do not share any component).*

Proof 1 *The first two points come from the acyclic property of the reachable relation $\rightarrow$. The third point is a consequence of hypothesis **Hyp2**. Let us suppose that there exists a common component α in the two branches ($\exists j, l, \alpha = b_j = c_l$), then because of the unicity of the path from i to α, we have $(b_1 \ldots b_j) = (c_1 \ldots c_l)$. Thus, $j = l$ and $b_k = c_k$ for all $k < l$. The fourth point then comes immediately.*



(a) $tr = i \cdot ((b_1 \ldots b_m) + \epsilon)$.

(b) $tr = i \cdot ((b_1 \ldots b_m) + (c_1 \ldots c_p))$ with $\forall l, k, b_l \neq c_k$.

(c) $tr = i \cdot ((a_1 \ldots a_n \cdot b_1 \ldots b_m) + (a_1 \ldots a_n \cdot c_1 \ldots c_p))$ with $\forall l, k, b_l \neq c_k$.

Figure 3: Transaction model

According to the proposition 3, a transaction can be graphically represented by the figure 3. Figure 3(a) shows a transaction made by a smart initiator. Figure 3(b) describes a transaction made by a non smart initiator without a common prefix. Figure 3(c) describes a transaction made by a non smart initiator with a common prefix. After the prefix, all the components are different. Note that the transaction Figure 3(b) is a particular case of Figure 3(c) in which the prefix word is empty: $(a_1 \ldots a_n) = \epsilon$.

3.3 Test classes revisited – Truncated transactions

A test class, as introduced by definition 3, is a set of single test classes $tc_1, \ldots, tc_n$ issued by n different initiators.

Definition 8 *Let $tc = \{(i_1, t_1) \ldots (i_m, t_m, t'_m)\}$ be a test class, the associated* PHYLOG *transactions are $phy(tc) = \{tr_1, \ldots, tr_m\} = \{phy(i_1, t_1), \ldots, phy(i_m, t_m, t'_m)\}$.*

The shared components in a set of transactions are then:

Definition 9 *Let $tc = \{tr_1, \ldots, tr_n\}$ be a set of transactions, the language associated to tc is defined by: $lg(tc) = lg(tr_1) \cup \ldots \cup lg(tr_n)$ and shared(tc) is the set of components involved in all transactions:*

$$shared(tc) = lg(tr_1) \cap \ldots \cap lg(tr_n)$$

We define the PHYLOG test classes as the sets of truncated transactions where the interference-free parts of transactions are translated as the empty word ϵ.

Definition 10 (PHYLOG test class) *A* PHYLOG *test class is a set of n transactions*

$$ptc = \{tr_1, \ldots, tr_n\}$$

such that $\exists tc = \{(i_i, t_i) \ldots (i_m, t_m)(i_{m+1}, t_{m+1}, t'_{m+1}) \ldots (i_n, t_n, t'_n)\} \in TC^P_\infty$ with

- $\forall k, hd(tr_k) = i_k$;
- $\forall j \leq m$, $tr_j = \text{trunc}(tc, phy(i_j, t_j))$ *where* $\text{trunc}(tc, i \cdot (a_1 \ldots a_n + \epsilon)) = i \cdot (a_1 \ldots a_p + \epsilon)$ *with* $p \leq n$ *and* $a_p \in shared(tc)$;
- $\forall m + 1 \leq j \leq n$, $\exists k \leq 3$, $tr_j = \text{trunc}_k(tc, phy(i_j, t_j, t'_j))$ *where*

$$\begin{array}{l}
\text{trunc}_1(tc, i \cdot (a_1 \ldots a_n + b_1 \ldots b_m)) = i \cdot (a_1 \ldots a_p + a_1 \ldots a_p) \\
\qquad \text{with } p \leq n \text{ and } a_p \in \text{shared}(tc) \text{ and } \forall k \leq p, b_k = a_k \\
\text{trunc}_2(tc, i \cdot (a_1 \ldots a_n + b_1 \ldots b_m)) = i \cdot (a_1 \ldots a_p + \epsilon) \\
\qquad \text{with } p \leq n \text{ and } a_p \in \text{shared}(tc) \text{ and } b_p \neq a_p \text{ and } \forall k, b_k \notin \text{shared}(tc) \\
\text{trunc}_3(tc, i \cdot (a_1 \ldots a_n + b_1 \ldots b_m)) = i \cdot (a_1 \ldots a_p + b_1 \ldots b_l) \\
\qquad \text{with } p, l \leq n \text{ and } a_p \in \text{shared}(tc) \text{ and } b_l \in \text{shared}(tc)
\end{array}$$

We note $\mathbb{T}rans$ the set of PHYLOG *test classes.*

Notations 2 *When handling truncated transactions generated by smart initiators, we will deliberately remove the second branch which is empty. For instance,* $\text{cpu}_1.\text{bus}$ *stands for* $\text{cpu}_1.(\text{bus} + \epsilon)$. *This contraction cannot be applied to truncated transactions generated by non smart initiators, as it would then be impossible to differentiate the prefix part.*

Example 5 *In P_1, $ptc = \{\text{cpu}_1.\text{bus}, \text{cpu}_2.\text{bus}\} \in \mathbb{T}rans$. Indeed,* $\text{cpu}_k.\text{bus}$ *can be obtained as the truncation of* $\text{cpu}_k \cdot (\text{bus.pcie} + \epsilon))$.

In architecture P_2, $ptc = \{\text{cpu}_1.\text{bus}_1, \text{dma}_1.(\text{bus}_1 + \epsilon)\} \in \mathbb{T}rans$. Indeed, let $tc = \{(\text{cpu}_1, \text{pcie}), (\text{dma}_1, \text{pcie}, \text{mem}_1)\}$ then $\text{cpu}_1.\text{bus}_1 = \text{trunc}(tc, \text{cpu}_1 \cdot (\text{bus}_1.\text{pcie} + \epsilon))$ *and* $\text{dma}_1.(\text{bus}_1 + \epsilon) = \text{trunc}_2(tc, \text{dma}_1.(\text{bus}_1.\text{pcie} + \text{bus}_2.\text{mem}_1))$.

However, in P_2, $\{\text{dma}_1.(\text{bus}_1 + \epsilon), \text{dma}_2.(\text{bus}_1 + \epsilon)\} \notin \mathbb{T}rans$. *Indeed,* $\text{dma}_k.(\text{bus}_1.X_1 + \text{bus}_1.X_2)$ *with $X_j \in \{pcie, L3\}$ can only be truncated with* trunc_1 *as* $\text{dma}_k.(\text{bus}_1 + \text{bus}_1)$. *And the test case* $\{\text{dma}_1.(\text{bus}_1.X_1 + \text{bus}_2.Y_2), \text{dma}_2.(\text{bus}_1.X_2 + \text{bus}_2.Y_2)\}$ *with $X_j \in \{pcie, L3\}$ and $Y_j \in \{\text{mem}_1, \text{mem}_2\}$ can only be truncated by* trunc_3 *as* $\text{dma}_k.\ (\text{bus}_1 + \text{bus}_2)$.

On the contrary, $ptc = \{\text{cpu}_1.\text{bus}_1, \text{dma}_1.(\text{bus}_1 + \epsilon), \text{dma}_2.(\text{bus}_1 + \epsilon)\} \in \mathbb{T}rans$. Indeed, the completion on the DMAs imposes to reach bus_2 *but this component will not be in the shared component as* cpu_1 *cannot reach it.*

3.4 Interference channel

We can now formally define the notion of interference channel. We distinguish two kinds of interference channels: the 1-interference channels which involve a single shared component, and the 2-interference channels which involve two shared components.

3.4.1 1-Interference channels

Definition 11 (1-Interference channel) *For an architecture $P = (\mathcal{C}, \rightarrow)$, a 1-interference channel is defined as*

$$(c, \bigcup_n TR_1^n)$$

with $c \in O \cup T$ and

$$TR_1^n = \bigcup \left\{ \begin{array}{l} \{tr_1, \ldots, tr_n\} \in \mathbb{T}rans \big| \\ \quad \forall i,\ tl(\mathrm{proj}_1(tr_i)) = c \wedge\ tl(\mathrm{proj}_2(tr_i)) = \{c, \epsilon\} \\ \quad \wedge\ \forall i \neq j, \\ \quad\ hd(tr_i) \neq hd(tr_j) \\ \quad\ \wedge\ \mathrm{pred}(\mathrm{proj}_1(tr_i)) \neq \mathrm{pred}(\mathrm{proj}_1(tr_j)) \end{array} \right\}$$

An interference channel ends with a shared component c. What happens after is irrelevant, as serialization occurs at that point and the transactions do not interfere with each other later on. We apply the same reasoning as in network calculus [16] when packets share a common path on several switches. This is known as the *pay burst only once* rule. Thus, interference only occurs on the first component shared by the n transactions. Note that this could be improved in several ways, such as by considering the component entailing the worst case delay instead of simply taking the first one, or by grouping successive components into a super-component.

Example 6 *Let us again consider the P_1 of Figure 1. There is a unique 1-interference channel with 11 combinations of transactions capable of occurring on the* bus*:*

$$\mathrm{bus}, \left\{ \begin{array}{l} \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{cpu}_2.\mathrm{bus}\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus}), \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus}), \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus}), \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus} + \mathrm{bus}), \mathrm{dma}_2.(\mathrm{bus} + \mathrm{bus})\} \end{array} \right\}$$

Example 7 *Let us consider P_2 of Figure 2 and let us focus on* bus_1 *(it is similar for* bus_2*). There are 28 combinations of transactions. The interesting parts are those featuring the* dma*. Either the second branch of the transaction reaches* bus_1*, or it is pruned (and replaced by ϵ). The notation* $\mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})$ *represents the two transactions* $\mathrm{dma}_2.(\mathrm{bus}_1 + \mathrm{bus}_1)$ *and* $\mathrm{dma}_2.(\mathrm{bus}_1 + \epsilon)$ *in order to reduce the length of the table below.*

$$\mathrm{bus}_1, \left\{ \begin{array}{l} \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{cpu}_2.\mathrm{bus}_1\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \mathrm{bus}_1)\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \epsilon)\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\}), \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\}), \\ \quad \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\}), \\ \quad \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\}, \\ \{\mathrm{cpu}_1.\mathrm{bus}_1, \mathrm{cpu}_2.\mathrm{bus}_1, \mathrm{dma}_1.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\}), \\ \quad \mathrm{dma}_2.(\mathrm{bus}_1 + \{\mathrm{bus}_1, \epsilon\})\} \end{array} \right\}$$

3.4.2 2-Interference channels

Focusing on 1-interference channels is unfortunately insufficient because of the double branches of transactions issued by non smart transactions. Indeed, those branches can conflict on two components: one per branch.

Example 8 *Let us consider the architecture of Figure 4. The two non smart initiators conflict on* b_1 *and* c_1*. None of them can be excluded in the analysis.*

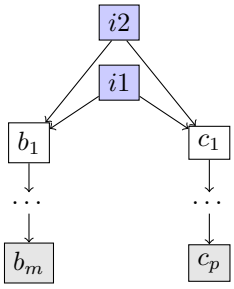


Figure 4: Architecture P_3

Thus, two components can be accessed in parallel by all transactions. We must then define 2-interference channels. There is no 3 or more-interference channels because the maximal number of branches per transaction is 2.

Definition 12 (2-Interference channel) *For an architecture* $P = (\mathcal{C}, \rightarrow)$*, a 2-interference channel is defined as*

$$(c_1, c_2, \bigcup_n TR_2^n)$$

with $c_1, c_2 \in O \cup T$*,* $c_1 \neq c_2$ *and*

$$TR_2^n = \bigcup \left\{ \begin{array}{l} \{tr_1, \ldots, tr_n\} \in \mathbb{T}rans \Big| \\ \quad \forall i,\ \ (tl(\mathrm{proj}_1(tr_i)) = c_1 \wedge tl(\mathrm{proj}_2(tr_i)) = c_2) \\ \quad \wedge \forall i \neq j,\ \ hd(tr_i) \neq hd(tr_j) \\ \qquad \wedge\ \ \mathrm{pred}(\mathrm{proj}_1(tr_i)) \neq \mathrm{pred}(\mathrm{proj}_1(tr_j)) \\ \qquad \wedge\ \ \mathrm{pred}(\mathrm{proj}_2(tr_i)) \neq \mathrm{pred}(\mathrm{proj}_2(tr_j)) \end{array} \right\}$$

Example 9 *In* P_2*, depicted in Figure 2, there is a unique 2-interference channel occurring on the buses.*

$$\mathrm{bus}_1, \mathrm{bus}_2, \{\ \{\mathrm{dma}_1 \cdot (\mathrm{bus}_1 + \mathrm{bus}_2),\ \mathrm{dma}_2 \cdot (\mathrm{bus}_1 + \mathrm{bus}_2)\}\ \}$$

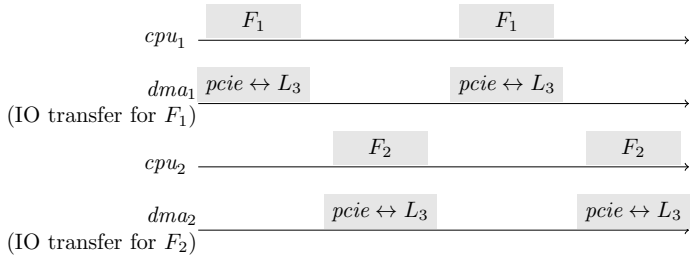


Figure 5: An execution scheme for architecture P_1

Example 10 *There is no 2-interference channel in the* P_1 *architecture. This means that, with our approach, analyzing the* severity *of the interference caused by the 11 transactions listed in example 6 is sufficient, compared to the* 7224 *test classes of the initiator target model (see example 3). This reduction comes from a better modeling of the platform and from symmetry properties. The* severity *of the interference is evaluated with respect to the expected behavior of the software functions hosted by the platform.*

To illustrate this, let us consider the following time-triggered execution scheme depicted in figure 5:
(1) cpu_1 (resp. cpu_2) hosts a software function F_1 (resp. F_2);
(2) memory mem_1 (resp. mem_2) is dedicated to F_1 (resp. F_2), meaning that F_1 never tries to access mem_2 and, conversely, F_2 never tries to access mem_1;
(3) dma_1 (resp. dma_2) manages input/output transfers of F_1 (resp. F_2);
(4) input/output data are stored in L_3;
(5) F_1 and F_2 are periodically scheduled in non-overlapping time windows;
and (6) dma_1 (resp. dma_2) is only activated by F_1 (resp. F_2).
Then, among the 11 transactions listed in example 6, only the second one ($\{\mathrm{cpu}_1.\mathrm{bus}, \mathrm{dma}_1.(\mathrm{bus}+\mathrm{bus})\}$) and the fifth one ($\{\mathrm{cpu}_2.\mathrm{bus}, \mathrm{dma}_2.(\mathrm{bus}+\mathrm{bus})\}$) can occur. The potential interference caused by the other nine transactions are avoided by the execution scheme. Therefore, to meet the certification requirements it is sufficient to evaluate the WCET of F_1 (resp. F_2) and the WCET of the dma_1 (resp. dma_2) transfers with the interference caused by the second (resp. fifth) transaction. These interference will be said to be acceptable *if these WCETs are smaller than the corresponding time windows planned by the execution model. Otherwise, they will be said to be* unacceptable.

4 Comparison between the initiator-target model and PHYLOG model

The PHYLOG model can be seen as the definition of equivalence classes for the *initiator-target* model.

Definition 13 (Interference-free test classes) *Some test classes listed in TC_∞^P do not lead to an interference channel:*

1. *Single test classes, as they do not generate any conflict,*
2. *Test classes tc such that $phy(tc) = (tr_1, \ldots, tr_n)$, $n > 1$ and $shared(phy(tc)) = \emptyset$, that is, no component is shared by the n transactions.*

Test classes satisfying one of these two rules are said to be interference-free.

Example 11 *Let us consider P_2 once again. The test class $\{(\mathrm{cpu}_1, \mathrm{mem}_1), (\mathrm{dma}_1, \mathrm{pcie}, \mathrm{L3})\}$ is interference-free: no component is shared by the two transactions (rule 2 of the definition). However, $\{(\mathrm{cpu}_1, \mathrm{mem}_1)$, $(\mathrm{dma}_2, \mathrm{pcie}, \mathrm{mem}_2)\}$ is not interference-free, since bus_1 is shared by the two transactions.*

Let us note that a transaction, as depicted in Figure 3, defines a partial order relation over the language for the transactions:

Definition 14 *For an architecture $P = (\mathcal{C}, \rightarrow)$ and a transaction $tr = i \cdot ((a_1 \ldots a_n \cdot b_1 \ldots b_m) + (a_1 \ldots a_n \cdot c_1 \ldots c_p))$, let us define the relation $<_{tr}$ over $lg(tr)$ as:*

- *if $(a_1 \ldots a_n) \neq \epsilon$, then $\forall i = 1 \ldots n-1, a_i <_{tr} a_{i+1}$*
- *if $(a_1 \ldots a_n) \neq \epsilon$, then $a_n <_{tr} b_1$*
- *if $(a_1 \ldots a_n) \neq \epsilon$ and if $(c_1 \ldots c_p) \neq \epsilon$, then $a_n <_{tr} c_1$*
- *$\forall i = 1 \ldots m-1, b_i <_{tr} b_{i+1}$*
- *if $(c_1 \ldots c_p) \neq \epsilon$, then $\forall i = 1 \ldots p-1, c_i <_{tr} c_{i+1}$*

$<_{tr}$ is the order generated by the oriented paths followed by the transaction.

Proposition 4 *Let $P = (\mathcal{C}, \rightarrow)$, $<_P$ is the partial order (see proposition 2), $tc = \{tr_1, \ldots, tr_n\}$ a PHYLOG test class, and $S = shared(tc)$ the set of components shared by all the transactions tr_i. Let*

$$min(S, <_P) = \{\alpha \in S \mid \forall \beta \in S, \text{ either } \alpha <_P \beta \text{ or } \neg(\beta <_P \alpha)\}$$

be the set of smallest components in S for $<_P$. Then

- *either $min(S, <_P) = \emptyset$,*

- *or* $min(S, <_P) = \{\alpha\}$ *and* $\forall tr_i, \forall \beta \in S, \alpha <_{tr_i} \beta$ *(meaning* α *is the first component crossed by all* tr_i*),*
- *or* $min(S, <_P) = \{\alpha, \beta\}$ *and* $\forall tr_i, \forall \gamma \in S, \alpha <_{tr_i} \gamma \ \vee \ \beta <_{tr_i} \gamma$ *(meaning* α *(resp.* β*) is the first crossed in some branches in which* β *(resp.* α*) is not involved, as* b_1 *and* c_1 *in Figure 4).*

Proof 2 *Let us consider* $tr_1 = i \cdot ((a_1 \dots a_m \cdot b_1 \dots b_p) + (a_1 \cdot a_m \cdot c_1 \dots c_q))$ *of tc with* $b_j \neq c_k$ *for all* j, k*. Remember that* $S = shared(tc) = lg(tr_1) \cap \dots \cap lg(tr_n)$*. Then* $S \subset \{a_1, \dots, a_m, b_1 \dots, b_p, c_1, \dots, c_q\}$*. Let us consider 5 cases:*

- *case 1:* $S = \emptyset$*. Then* $min(S, <_P) = \emptyset$*. Meaning tc is interference-free.*
- *case 2:* $S \cap \{a_1, \dots a_m\} \neq \emptyset$*. Then* $\exists k, min(S, <_{tr_1}) = \{a_k\}$*. Moreover,* $min(S, <_P) = min(S, <_{tr_1}) = \{a_k\}$*.*
- *case 3:* $S \subset \{b_1, \dots b_p\}$*. Then* $\exists k, min(S, <_{tr_1}) = \{b_k\}$*. Then* $min(S, <_P) = min(S, <_{tr_1}) = \{b_k\}$*.*
- *case 4:* $S \subset \{c_1, \dots c_q\}$*. Then* $\exists k, min(S, <_{tr_1}) = \{c_k\}$*. Then* $min(S, <_P) = min(S, <_{tr_1}) = \{c_k\}$*.*
- *case 5:* $S \subset \{b_1, \dots, b_p, c_1, \dots c_q\}$ *and* $S \cap \{b_1, \dots b_p\} \neq \emptyset$ *and* $S \cap \{c_1, \dots c_q\} \neq \emptyset$*. Then* $\exists k, l, min(S, <_{tr_1}) = \{b_k, c_l\}$ *with* $\neg(b_k <_{tr_1} c_l)$ *and* $\neg(b_k <_{tr_1} c_l)$*. Then* $min(S, <_P) = min(S, <_{tr_1}) = \{b_k, c_l\}$*.*

Example 12 *For* P_2 *and its* $\{(\mathrm{dma}_1, \mathrm{mem}_1, \mathrm{pcie}), (\mathrm{dma}_2, \mathrm{mem}_2, \mathrm{pcie})\}$ *test class, the associated transactions are*

- $tr_1 = \mathrm{dma}_1 \cdot (\mathrm{bus}_1 \cdot \mathrm{pcie} + \mathrm{bus}_2 \cdot \mathrm{mem}_1)$
- $tr_2 = \mathrm{dma}_2 \cdot (\mathrm{bus}_2 \cdot \mathrm{mem}_2 + \mathrm{bus}_1 \cdot \mathrm{pcie})$

$S = \{\mathrm{bus}_1, \mathrm{bus}_2, \mathrm{pcie}\}$*, and* $min(S, <_P) = \{\mathrm{bus}_1, \mathrm{bus}_2\}$*.*

Definition 15 (Relation $\equiv$) *Let us define the relation* $\equiv$ *on the Brindejonc et al. test classes. Let* tc_1 *and* tc_2 *be two test classes. Let* $S_1 = shared(phy(tc_1))$ *and* $S_2 = shared(phy(tc_2))$*, then*

$$tc_1 \equiv tc_2 \iff \min(S_1, <_P) = \min(S_2, <_P) \wedge (S_1 = \emptyset \vee hd(tc_1) = hd(tc_2))$$

Proposition 5 *The relation* $\equiv$ *is an equivalence relationship.*

Proof 3 $\equiv$ *is reflexive. Indeed, let tc be a test class,* $\min(shared\ (phy(tc)), <_P)$ *is defined in a unique way.*

$\equiv$ *is symmetric, because we only handle sets.*

$\equiv$ *is transitive: if* $\min(S_1, <_P) = \min(S_2, <_P)$ *and* $\min(S_1, <_P) = \min(S_3, <_P)$ *then* $\min(S_2, <_P) = \min(S_3, <_P)$*. Same for* $hd(tc_i)$*.*

Proposition 6 *The* PHYLOG *interference channels are a representative of the* $\equiv$ *relation classes. More precisely, let* $tc = \{(i_1, t_1) \dots (i_m, t_m, t'_m)\}$*, let* $S = shared(phy(tc))$*.*

- *if* $min(S, <_P) = \emptyset$*, there is no interference channel in* PHYLOG*,*
- *if* $min(S, <_P) = \{c\}$*, the associated interference channel in* PHYLOG *is a 1-interference channel in* $c, \cup_n TR_1^n$*, i.e.* $tr = \{tr_1, \dots, tr_m\} \in TR_1^m$ *with* $hd(tr) = hd(tc)$*;*
- *if* $min(S, <_P) = \{c_1, c_2\}$*, the associated interference channel in* PHYLOG *is a 2-interference channel* $c1, c2, \cup_n TR_2^n$*, i.e.* $tr = \{tr_1, \dots, tr_m\} \in TR_2^m$ *with* $hd(tr) = hd(tc)$*.*

Proof 4 *Case interference-free channel: all these test classes are associated to the empty set of* PHYLOG*.*

Case $min(S, <_P) = \{c\}$ *(resp.* $= \{c_1, c_2\}$*): we apply the truncation of definition 10 where the transactions stop at* $a_p = c$ *(resp.* $a_p = c_1$ *and* $b_l = c_2$*) and then* $\{tr_1, \dots, tr_m\}$ *is in* TR_1^m *(resp.* TR_2^m*).*

5 Experiments

In this section, we first illustrate the IDP code supporting the interference channels computations. We then provide some experiments with IDP.

5.1 Coding the initiator target model

SI, *NSI*, T and O are represented as types. $\rightarrow$ is a predicate with two parameters.

Code 1
```
type SInitiator
type NSInitiator
type Initiator contains SInitiator, NSInitiator
type Reactive
type Target
type Component contains Reactive, Target, Initiator
```

A predicate *Path* computes all paths from *SI* to T. For our modeling, we chose to hard code the maximal length of a path. Thus, if a path is shorter, we use *NULL* to complete the path. For instance, in P_1, if the maximal length is 4, the path from CPU_1 to mem_1 is represented as *Path(*CPU_1*, interconnect,* mem_1*, NULL) = true*.

Code 2
```
!i[Initiator]: !x1[Component]: !x2[Component]:
!x3[Component]: Path(i,x1,x2,x3) <-
  (Edge(i,x1) & Edge(x1,x2) & Edge(x2,x3)).
```

A predicate *SingleTestCase* computes the single test cases. Again, the length is imposed to 3, because of the non smart initiator. The test case $CPU_1 \rightsquigarrow mem_1$ is represented as *SingleTestCase(*CPU_1*,* mem_1*, NULL) = true*.

Code 3
```
!i[NSInitiator]: !t1[Target]: !t2[Target]:
SingleTestCase(i,t1,t2) <-
 ?x[Component]: ?z[Component]: ?v[Component]:
 ?y[Component]: ?h[Component]: ?l[Component]:
 (Path(i,x,t1,z) | Path(i,x,z,t1) | Path(i,t1,z,v))
 & (Path(i,y,t2,h) | Path(i,y,h,t2)
 | Path(i,t2,h,l)).
```

Finally, there is one predicate per TC_n^P. For instance, test cases of size 2 are coded by *CoupleTestCase*.

Code 4
```
!i1[NSInitiator]: !t1[Target]: !t2[Target]:
!i2[NSInitiator]: !t3[Target]: !t4[Target]:
CoupleTestCase(i1,t1,t2,i2,t3,t4) <-
 (SingleTestCase(i1,t1,t2) &
 SingleTestCase(i2,t3,t4) & (i1 <i2)).
```

5.2 Coding the PHYLOG model

We compute an intermediate predicate to combine two transactions composed of a single branch each. *Trans2*, still for a maximal of 3 components in a path, is defined below.

Code 5
```
// when the shared component is at the end
!x[Component]: !i1[Initiator]: !i2[Initiator]:
!x1[Component]: !x2[Component]:
!x3[Component]: !x4[Component]:
Trans2(x,i1,x1,x2,x,i2,x3,x4,x) <-
 (Path(i2,x3,x4,x) & Path(i1,x1,x2,x)
  & (i1 < i2) & (x2~=x4)).
```

There is one predicate per TR_1^n. For instance, TR_1^2 is coded by combining the two branches of a pair

of transactions.

Code 6
```
// case 1 SI and 1 NSI
!x[Component]: !i1[SInitiator]: !i2[NSInitiator]:
!x1[Component]: !x2[Component]: !x3[Component]:
!z1[Component]: !z2[Component]: !z3[Component]:
1Interf2(x,i1,x1,x2,x3,x4,NULL,NULL,NULL,
       NULL,i2,z1,z2,z3,z4,z1,z2,z3,z4) <-
 Trans2(x,i1,x1,x2,x3,x4,i2,z1,z2,z3,z4).
```

There is also one predicate per TR_2^n. The code for TR_2^2 is given below.

Code 7
```
!x1[Component]: !x2[Component]: !i1[NSInitiator]:
!i2[NSInitiator]: !y1[Component]: !y2[Component]:
!y3[Component]: !y4[Component]: !y5[Component]:
!y6[Component]: !z1[Component]: !z2[Component]:
!z3[Component]: !z4[Component]: !z5[Component]:
!z6[Component]:
2Interf2(x1,x2,i1,y1,y2,y3,y4,y5,y6,
        i2,z1,z2,z3,z4,z5,z6) <-
 (Trans2(x1,i1,y1,y2,y3,i2,z1,z2,z3)
  & Trans2(x2,i1,y4,y5,y6,i2,z4,z5,z6)
  & (x1 < x2)).
```

5.3 Some results

IDP computes the predicates of the previous examples in less than a second. Increasing the number of smart initiators, non-smart initiators, targets, or intermediary components does not appear to increase the execution time. Even when modeling a Kalray MPPA [8] compute cluster, composed of 16 cores, 32 intermediary components, and 16 targets, resolution of those predicates (exposing a total of 1920 interference channels) is still completed in sub-second times.

To ensure their correctness, we have also used IDP to compute the $\equiv$ relation classes of the aforementioned examples, and found them to be compliant with the propositions made in this paper.

6 Related Work

Interference analysis in multi-core processors has received significant attention in recent years. A first class of these works focuses on the impact of shared hardware resource contention on the execution time of software application hosted by the processor. For instance, [9] considers a multi-core architecture composed of a single bus providing access to a shared memory, and it proposes a method to determine an upper bound on the number of bus requests that software tasks can generate in a given time interval. Both [5] and [12] focus on measurement techniques based on dedicated stressing benchmarks and hardware monitors to characterize the architecture and the shared resources that can cause interference between software applications.

A second class of works focuses on methods to avoid interference. For instance, [2] proposes a contention-free execution framework to execute automotive software application on many-core platforms. [19] proposed a similar approach which relies both on a development work-flow, and the use of an execution model defined as a set of rules to be followed by the designer and asserted through the run-time in order to enforce specific behaviors. Both [2] and [19] target a TDMA execution model, and use a Constraint Programming formulation to find an optimal time-triggered schedule on each core.

In order to tackle multi-core aeronautics certification-related issues, several projects have been funded. One of the first was MULCORS [13], which clearly identified the need to change and adapt the current certification standard. Since then, several attempts at precisely defining such new recommendation have been done, such as the Multi-core Certification Review Item (MCP-CRI) [11]. In other parts of the MCP-CRI, [15] proposed definitions for interference channels, interference sources, and interference targets, and they proposed a process to reduce the number of interference. A more recent work proposed by [1] tried to adapt the MCP-CRI certification objectives to COST MCP architectures. For that purpose, they showed that the MCP-CRI objectives can be grouped into three high level principles: (1) determining the final configuration, (2) managing interference channels, and (3) verifying the use of shared resources. They

showed, through a particular case study (the Freescale P4080 processor), that the second objective (managing interference channels) highly depends on detailed information about the behavior of the resources. And they showed that predicting interference on a COTS multi-core architecture is a very challenging task because of the amount of required information. A way to help the certification application to master the complexity of the architecture is then to use a formal model of the architecture and a formal analysis method to explore the set of interference channels. Such is the aim of our contribution.

7 Conclusion

In this paper, we have formally defined the *initiator-target model* and compared it with the PHYLOG approach. Our representation requires more details on the internal of the platform but offers a more practical size description. Our work was supported with IDP tool.

In the future, we will apply our model to other multi-cores and extend our model to many-core platforms. We will also measure the gains of going deeper in the description of the architecture.

References

[1] Irune Agirre, Jaume Abella, Mikel Azkarate, and Francisco Cazorla. On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures. In *12th IEEE International Symposium on Industrial Embedded Systems (SIES'17)*, 2017.

[2] Matthias Becker, Dakshina Dasari, Borislav Nicolic, Benny Åkesson, Vincent Nélis, and Thomas Nolte. Contention-free execution of automotive applications on a clustered many-core platform. In *28th Euromicro Conference on Real-Time Systems*, July 2016.

[3] Pierre Bieber, Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Claire Pagetti, Olivier Poitou, Thomas Polacsek, Luca Santinelli, and Nathanaël Sensfelder. A model-based certification approach for multi/many-core embedded systems. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[4] Pierre Bieber, Frédéric Boniol, Guy Durrieu, Olivier Poitou, Thomas Polacsek, Virginie Wiels, and Ghilaine Martinez. MIMOSA: Towards a model driven certification process. In *Proc. 8th Int. Congress on Embedded Real Time Software and Systems (ERTS'16)*, 2016.

[5] Jingyi Bin, Sylvain Girbal, Daniel Gracia Perez, Arnaud Grasset, and Alain Merigot. Studying co-running avionic real-time applications on multi-core cots architectures, 02 2014.

[6] Vincent Brindejonc and Anthony Roger. Avoidance of dysfunctional behaviour of complex cots used in an aeronautical context. In *19eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2014.

[7] Certification Authorities Software Team. Multi-core Processors - Position Paper. Technical Report CAST 32-A, November 2016.

[8] Kalray Corporation. *The MPPA hardware architecture*, 2012.

[9] Dakshina Dasari and Vincent Nelis. An analysis of the impact of bus contention on the wcet in multicores. In *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, HPCC '12, pages 1450–1457, Washington, DC, USA, 2012. IEEE Computer Society.

[10] Broes de Cat, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Predicate logic as a modelling language: The IDP system. *CoRR*, abs/1401.6312, 2014.

[11] EASA (European Aviation Safety Agency). The Use of Multi-Core Processors in Safety-Critical Applications - CRI, 2016.

[12] Sylvain Girbal, Jingyi Bin, Daniel Gracia Perez, and Alain Merigot. Using monitors to predict co-running safety-critical hard real-time benchmark behavior. In *Conference on Information and Communication Technology for Embedded Systems (ICITES'14)*, 01 2014.

[13] Xavier Jean, Marc Gatti, Guy Berthon, and Marc Fumey. MULCORS-Use of Multicore Processors in airborne systems. *European Aviation Safety Agency, Industrial report December*, 2012.

[14] Xavier Jean, Laurence Mutuel, and Vincent Brindejonc. Assurance methods for cots multi-cores in avionics. In *35th Digital Avionics Systems Conference (DASC'16)*, 2016.

[15] Xavier Jean, Laurence Mutuel, Didier Regis, Hélène Misson, Guy Berthon, and Marc Fumey. White Paper on Issues Associated with Interference Applied to Multicore Processors, 2016. Retrieved from `http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/SDS_DO005_White_Paper.pdf`.

[16] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[17] Laurence Mutuel, Xavier Jean, and Vincent Brindejonc. Investigation of error types associated with failures in multicore processors. In *20eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2016.

[18] Laurence Mutuel, Xavier Jean, Vincent Brindejonc, Anthony Roger, Thomas Megel, and E. Alepins. Assurance of Multicore Processors in Airborne Systems, 2017.

[19] Quentin Perret, Pascal Maurère, Éric Noulard, Claire Pagetti, Pascal Sainrat, and Benoît Triquet. Temporal isolation of hard real-time applications on many-core processors. In *22th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16)*, April 2016.

[20] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions Embedded Computing Systems*, 7(3):36:1–36:53, May 2008.

[21] Reinhard Wilhelm and Jan Reineke. Embedded systems: Many cores - many problems. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 176–180, 2012.