A model-based certification approach for multi/many-core embedded systems

Pierre Bieber, Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Claire Pagetti, Olivier Poitou, Thomas Polacsek, Luca Santinelli, Nathanaël Sensfelder

A model-based certification approach for multi/many-core embedded systems

Pierre Bieber, Frédéric Boniol, Youcef Bouchebaba, Julien Brunel
Claire Pagetti, Olivier Poitou, Thomas Polacsek, Luca Santinelli, Nathanaël Sensfelder
ONERA-Toulouse, France

***Abstract*—This article presents the first year results of the PHYLOG project that aims at providing a model-based certification framework for aeronautics systems designers when developing multi/many-core-based architectures. After a brief reminder of the certification objectives, we present an overview of the PHYLOG approach. We then detail two points of the methodology: (1) a certification oriented meta-model for multi- and many-core platforms, and (2) an interference prediction method based on this meta-model. This method is then illustrated on a sub-part of the many-core Kalray MPPA®-256.**

I. INTRODUCTION

Certification activities consist in providing detailed documentation and justifications that explain why the development of a specific product is trustworthy and fulfills a given standard's requirements. Such a comprehensive documentation not only contains the results, but also the input data, the hypotheses, the techniques applied, etc. This process is well covered by the current aeronautics practices. However, for the next generation of multi/many-core-based architectures, the means of compliance will evolve due to architecture specifics.

A. Context

The last decade has seen the emergence of multi-core and many-core architectures, i.e. chips integrating several cores interconnected by either a shared bus (for multi-core processors) or a networks on chip (for many-core processors). Although these architectures may allow a huge gain in terms of performance, they also face important challenges to their integration in safety critical environments. As an example, due to the intensive resource sharing and lack of documentation, it is very difficult to ensure time predictability [WEE+08], [WR12], one of the key elements of certification expectation.

In order to tackle multi-core aeronautics certification-related issues, several projects have been funded. One of the first was MULCORS [JGBF12] which clearly identified the need of changing and adapting the current certification standard. Since then, several attempts at precisely defining such new standard have been done, such as the Multi-core Certification Review Item (MCP-CRI) [EAS16], the CAST Position Paper #32 [CAS14], and the FAA white paper [JMR+16].

B. Objectives

The objective of the article is to explore a model-based approach, to help both the applicant to answer the MCP-CRI requirements and the certification authority to assess the arguments provided by the applicant. Indeed, the standards mentioned previously describe high level requirements, but no means with which to ensure them nor ways to demonstrate the compliance of an integrated system (multi-core processor + applicative software). Thus, as is, the proposed approach does not intend to replace the MCP-CRI, but instead to offer a methodological framework to assist certitication. This work was done within the PHYLOG DGAC project and the results are those of the first year.

In previous works [BBD+16], [Pol16], we proposed a model-based framework to simplify the certification of aeronautics systems, to cope with inflation of documentation, to improve the coverage of requirements, and to ease the use of formal methods as means of verification. In PHYLOG, we reuse and improve those ideas to specifically address multi/many-core issues.

In the sequel, we present the main objectives exposed by the new standards (section II). We present, in section III, an overview of the methodology developed during PHYLOG project. We then detail two points of the PHYLOG methodology: (1) a certification oriented modeling language for MCP platforms (section IV), and (2) an interference prediction method based on the formal modeling presented above. A detailed example on how to apply the modeling language and the interference prediction method is given in section VI.

II. REMINDER OF CERTIFICATION OBJECTIVES

The Multi-core Certification Review Item (MCP-CRI) [EAS16], also published as the CAST-32A position paper, provides a set of guidances for software planning and verification on multi-core chips, with a particular emphasis on timing considerations and error handling.

A. Main objectives of MCP-CRI

These guidances are structured in 5 high level objectives.

a) Software planning: According to this objective, the applicant shall provide a model of the multi-core architecture. He has to identify the specific processor, the number of active cores, the software architecture, the dynamic software features, etc. He also has to argue whether the platform provides robust partitioning or not. And finally he has to detail the methods and the tools used for the development and verification of all the software components hosted on the platform (including hypervisors and operating systems).

b) Planning and setting resources: The second objective deals with the *final configuration* (i.e., the stable configuration reached after the boot step). The applicant has to detail all the configuration settings, including the execution frequency of the activated cores, which of the peripheral devices are activated, how the caches, memories, and interconnects are configured and allocated, etc. Consequently, the applicant shall identify the shared resources and shall describe a usage domain for each of them (how the resources are shared and how to prevent resource capabilities from being exceeded). Complementarily, the applicant shall also argue that the critical configuration settings are static and are protected against inadvertent changes at run-time.

c) Interference Channels and resource usage: This is the main objective. Due to resource sharing, coupling exists at the platform level, which can cause interferences between the applications. These interferences can lead to unexpected delays or loss of data. In order to prevent potential unpredictable behaviors, the applicant has to identify all the interference channels in the final configuration, and he has to argue that they are properly mitigated by adequate means. Afterwards, the applicant shall argue that the resource demand does not exceed the resource availability in the final configuration (when taking into account the resource usage and the mitigation means).

d) Software verification: Due to interference channels, the applications hosted by a multi-core platform could suffer from unexpected delays. The applicant shall thus verify that all the software components (including the operating systems and hypervisors) operate correctly and do not miss any deadlines when running in parallel in the final configuration. The MCP-CRI distinguishes two cases: (a) either the platform provides robust partitioning mechanisms, or all the interferences are properly avoided or mitigated, then the verification and the timing analysis can be done for each application in isolation; (b) otherwise, the platform and all its application components have to be considered in the same verification step. In the latter case, the applicant is not allowed to proceed incrementally.

e) Error detection and handling: Due to the complexity of executing in parallel several software components competing for the same resources, specific errors and failures may happen (e.g., memory violation). They need to be detected and handled at run-time. For that purpose, additional mechanisms may need to be developed and verified. This last objective requires the applicant to detail the error handling solution integrated in the platform (including the use of a *safety net* external to the multi-core chip), and to argue that all the errors and failure conditions are properly managed.

B. Refinement of the CRI objectives

Previous works have revisited the MCP-CRI certification objectives, in particular the timing interferences. The authors of [JMR+16] propose more detailed definitions for interference channels, interference sources, and interference targets, and they propose a process to reduce the number of interferences. The aim of [AAAC17] is to adapt the MCP-CRI certification objectives to COST MCP architectures. For that purpose, they group these objectives into three high level principles:

- *Determining the final configuration*: determining which configuration is the final one, and showing that it is protected against inadvertent changes at run-time;
- *Managing interference channels*: identifying all the interference channels in the final configuration, defining the means to either avoid interference by design or to upper-bound them such that the deadlines of the software components are satisfied;
- *Verifying the use of shared resources*: showing that the software components do not exceed the use of available resources.

The idea of [AAAC17] is that following these three high level principles should help answering the MCP-CRI objectives. We also follow this idea. In the PHYLOG approach, we intend to cover all the MCP-CRI objectives, including the safety concern. We extend this classification into four high level objectives, as shown in the table I. Our thesis is that answering these four objectives in a correct, complete, and consistent way should ease the providing of certification files compliant with the MCP-CRI requirements.

Conf	*Modeling the final configuration*, i.e., modeling the platform, including the software and the hardware components, the additional mitigation means, the execution model, etc. The model shall describe normal and abnormal behaviors
Interf1	*Determining the interference channels*, i.e., identifying all the residual interference channels while considering the additional mitigation means added to avoid or limit conflicts on shared resources
Interf2	*Arguing the interference channels are innocuous*, i.e., arguing that the interference channels identified by objective **Interf1** either never happen, have a negligible effect, or are upper-bounded such that all the deadlines of all the software components are still satisfied and no data is lost
Error	*Arguing errors and failures are managed*, i.e., arguing that abnormal behaviors are properly mitigated by internal means or by an external safety net, including that the configuration settings are protected against inadvertent changes at run-time

TABLE I
PHYLOG REFINED CERTIFICATION OBJECTIVES

III. MODEL-BASED CERTIFICATION APPROACH: OVERVIEW

Following this thesis, the PHYLOG project aims at building a reference framework for the certification of multi- and many-core architectures. The goal is to formalize certification requirements and to provide a way to build coherent and readable augmentations. According to the classification table I, the framework and its associated methodology involve 3 steps:

1) a modeling step (answering **Conf**);
2) an analyzing step (answering **Interf1**);
3) an argumentation step (answering **Interf2** and **Error**).

A. The modeling step

Providing a complete and consistent description of an architecture is the first certification issue. A common way

to achieve this goal is to describe the equipment in a textual manner. However, such method lacks formalization and verification means. The first idea suggested by PHYLOG is to follow a formal language-based approach. The proposed language, called *PML* (for PHYLOG Modeling Language) shall offer the capabilities to model the fundamental concepts of multi/many-core processors involved in the MCP-CRI (e.g., shared resources, hardware configuration, interference channels, mitigation means, etc.). In that sense, it differs from other architecture languages such as AADL or SysML, for it is not a design oriented language, but a certification oriented language. PML is based on 4 concerns (depicted in the MCP-CRI):

1) hardware platform (e.g., cores, MMU, interconnect, etc.),
2) software platform (e.g., OS, hypervisor, etc.),
3) applicative software,
4) deployment.

To be able to fully support the users – both the industrial applicant and the certification authority –, we need reasoning capabilities on this high level language. To this end, PML is translated to a logical framework, called WEIRD [BBD+16], that offers the expected reasoning capabilities.

PML, WEIRD, and the PML implementation in WEIRD are presented in section IV.

B. Interference prediction and interference analysis

As said previously, interferences between applications are one of the main concerns of the MCP-CRI. The certification objective **Interf1** requires the ability to predict all the possible interferences in the final configuration, and to show that they are innocuous with respect to the applications requirements. This activity is decomposed into two sub-steps: (a) interference prediction, and (b) interference analysis.

1) Interference prediction: The aim of this first sub-step is to predict all the "residual" interference channels. By "residual", we mean the interference channels which are not avoided by partitioning means (such as an hypervisor). We consider three types of interference: (a) hardware interference, i.e., conflict when accessing the same hardware component (e.g., interconnect, memory, etc.), logical interference, i.e., conflict when accessing the same logical component (e.g., semaphore, OS service, driver, etc.), and (c) data interference, i.e., conflict when reading or writing the same data. To predict those conflicts, we follow the transaction-driven approach proposed in [JMR+16]. From a description of the transaction domains, i.e., the description of the paths followed by transactions issued by software and hardware components through the architecture, we compute all the situations where two (or more) transactions can overlap on the same resource. Such a situation leads to a potential interference.

One of the benefits of using a formal logical framework is to be able to compute all these potential interferences exhaustively and automatically. This point will be detailed in section V.

2) Interference analysis: Once an interference is predicted, the interference analysis aims at assessing if the interference can really happen, and what are its effects with respect to application requirements. A final configuration is said to be interference-free if all of the residual interferences are negligible.

Note that the prediction and the analysis sub-steps shall be done for both the normal conditions (without any failure), and the abnormal conditions (hardware failure). The prediction sub-step can be done automatically by using a constraint programming method. However, the second sub-step is discharged to specific analysis methods that are outside of the PHYLOG framework.

C. Argumentation

Argumentation is the third key issue required by objectives **Interf2** and **Error**. When inquiring the certification of a system, the inquirer must provide a certification file. According to [MoD03], this should be "A reasoned, auditable argument created to support the contention that a defined system will satisfy the requirements". The exact nature of the elements is not detailed, but graphical representation or models are increasingly becoming part of this certification file. For the safety aspects, [MoD07] explicitly requires safety cases, from which our work is inspired.

The goal of the PHYLOG Argumentation step is to represent graphically the different means of compliance used to justify the satisfaction of the requirements. The Argumentation step organizes the various elements (formal and informal) that contribute to the justification of the requirements. The argumentation step of PHYLOG is inspired by GSN [KW04]. It is also inspired by Toulmin works for the underlying principles [Tou03], and from existing work on assurance cases in the aeronautical domain [Hol15], [Pol16].

A generic argumentation step relies on the following concepts (figure 1):

- *Claim*: the property to be justified (will often link to a requirement),
- *Evidence*: the facts that will be used to justify the claim (analysis results, test results, expert knowledge, bibliographical reference...),
- *Strategy*: combination of different evidences in order to justify a claim, this is the model counter-part of "Mean of Compliance".
- *Usage Domain*: domain on which the method is usable
- *Rationale*: justifies the use of the method in this particular context

Argumentation patterns are then proposed as a partial instance of this generic step, specifying subconcepts, known entities, and their necessity status depending on the strategy or strategy family. Argumentation patterns can be of different natures: generic (for example, "Using a tool" that will make it mandatory to explicit the usage domain and add a corresponding support to show its respect) or domain specific (like "Showing that the Worst Case Delay caused by an interconnect interference is less than X"). An argumentation is then built as

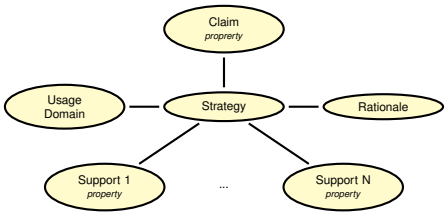

Fig. 1. The generic argumentation step

a chain of argumentation steps: one claim of a level becomes support for the next one.

D. Certification requirements revisited

Another benefit of the proposed model-based approach is that the certification objectives table I can be associated to more precise certification requirements expressed and checked on the models. These requirements are classified in three families, each of which is divided in two sub-groups:

- *Consistency requirements*:
  - ConsReq1: the architectural model provided at the Modeling step shall satisfy consistency rules such as "all applicative software shall be allocated to a unique core", "all hardware resources shall be described by exactly one normal behavior and at least one failure behavior", etc.
  - ConsReq2: the argumentation model provided at the Argumentation step shall satisfy consistency rules, with respect to the argumentation patterns, such as "all argumentation steps using the "verification tool" strategy shall exhibit the usage domain of the tool", etc.
- *Completeness requirements*:
  - ComplReq1: the behaviors associated with each component from the architecture model shall be validated by a corresponding claim in the argumentation model. For instance, if an interconnect device is modeled as a set of parallel lines without interference, this description shall be considered as an assumption in the modeling step, which shall be validated by an appropriate justification in the argumentation step. This justification can be supported either by technical documents from the MCP manufacturer, by experiments, or by expert judgements, etc.
  - ComplReq2: all the residual interference channels identified by the interference prediction step shall be associated with a claim in the argumentation model, arguing that either the interference never happens or that it has no undesirable effect.
- *Correctness requirements*:
  - CorrReq1: the method used to predict residual interference channels shall be safe, meaning that if an interference channel could happen, then it must be predicted.
  - CorrReq2: all the interference analysis discharged to external means shall be safe, meaning that if the external means claim that the interference never happens or has no undesirable effect, then it must always be the case.

The ConsReq1 and ConsReq2 lists given above are not exhaustive. Those lists may depend on the under consideration MCP and can be negotiated between the industrial applicant and the certification authority. Nevertheless, once decided, the consistency and completeness requirements can be proven automatically using the WEIRD logical framework.

IV. MCP MODELING

As expressed in the previous section, modeling is the first key issue. The idea is to gather in an understanding model all the knowledge about the MCP configuration needed for the certification purposes. Such an approach provides a way to "query" and reason about the resulting information base. For that purpose, we have defined a certification-oriented modeling langage (called PML) for MCP platforms.

A. PML: main concepts

The aim of the modeling phase is to capture and to formalize the concepts needed for the certification argumentation, and, more precisely, the concepts involved in the arguments showing that (i) the interferences due to resource sharing either are properly mitigated, or have a negligible effect on the behavior of the software applications.

As suggested by MCP-CRI, PML is organised in 4 chapters: (1) hardware platform, (2) software platform, (3) applications, and (4) integration.

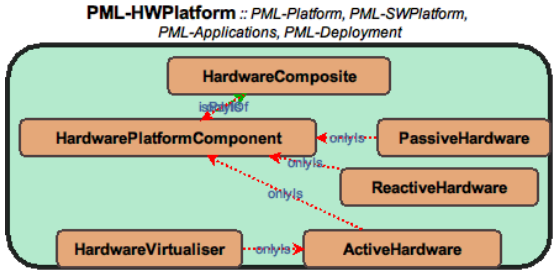

Fig. 2. Meta-model of the hardware platform chapter

1) Hardware platform chapter: Figure 2 presents the concepts involved in the hardware platform chapter. A HW Platform is a tree structure composed of *Hardware Platform Components* (noted HWP component for short). Each HWP component is either a terminal component, or an element which itself breaks down into sub-elements. For example, many-core processors are generally composed of processing-clusters, themselves composed of processing-cores, which are in turn broken down into data-cache, instruction-cache, processing-unit, etc. We call *Hardware Composite* (noted HC) a HWP component that contains sub-components.

In this hierarchy, the terminal components are of three types:

- *Active Hardware* (noted AHW). An AHW component is a hardware component that can issue transactions (such as *memory access*). By *active*, we mean that the component is able to generate transactions on its own initiative. Examples of AHW are DPAA (Data Path Acceleration Architecture), DMA, etc.
- *Reactive Hardware* (noted RHW). A RHW component can only transmit transactions issued by active components. By *reactive*, we mean that the component is not able to issue transactions on its own initiative. Examples of RHW are interconnect, memory controller, etc.
- *Passive Hardware* (noted PHW). A PHW component can not issue nor transmit transactions. It can only be targeted by transactions. Examples of PHW are memory banks, cache ways, etc.

Active and passive components correspond to what authors of [JMR+16] have called *initiator* and *target* components. More precisely, their definition of a *smart initiator* (i.e., an *initiator* with memory) matches the notion of *Hardware Composite* components explained a few paragraphs below, whereas the *non-smart initiators* (i.e., *initiators* without memory) are indeed active components.

In addition to these hardware elements, we introduce a more specific hardware component whose aim is to virtualize hardware resources (concept *Hardware Virtualiser* (noted HV)). Examples of HV component are MMUs, which allow the virtualisation of memory into memory partitions.

Note that from this classification, the Freescale CoreNet Coherency Fabric involved in the P40 and T10 families is considered to be a *Hardware Composite* component. It is composed of two elementary components: (1) a crossbar considered to be a *Reactive Hardware* component, as it can only transmit transactions between the cores and the peripheral components, and (2) a coherency fabric considered to be an *Active Hardware* component, as it can initiate coherency transactions between the caches involved in the cores.

This distinction between active, reactive, and passive components will be useful for the *interferences* computation. An interference is an event caused by a "*collision*" or an "*overlap*" on the same shared HW component between two transactions issued by two active components and propagating through reactive components. The shared component where the interference occurs can be either active, reactive or passive.

Note that, in order to be as generic as possible, that is, to be able to address existing or future multi-core and many-core processors, the meta-model of the HW platform chapter does not offer business concepts such as "*core*", "*interconnect*", etc. Introducing business concepts would lead (1) to a great number of concepts, and (2) to a lack of genericity. We chose to capture only abstract concepts from which the concrete components can be modeled. For instance, as explained previously, a memory will be described as a *Hardware Composite* component, composed of a controller (considered as reactive), and several banks (considered as passive).

2) Software platform chapter: Figure 3 presents the meta-model of the software platform chapter. This chapter describes

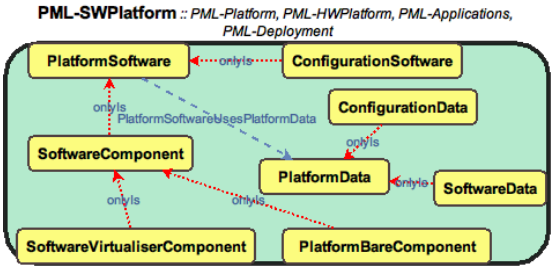

Fig. 3. Meta-model of the software platform chapter

the software architecture involved for managing the platform. It does not include applicative software. A software platform contains software elements (called *Platform Software* component), refined into (1) *Configuration Software* involved in the boot phase to configure hardware resources, and (2) *Software Components* involved at run-time. Software components are either bare components, directly running on hardware resources, or virtualiser components whose aim is to create and to manage virtual resources (such as compute partition).

Note that *Platform Software* and *Configuration Software* are abstract classes, meaning that any platform model cannot contain instances of these classes. Software element involved in the platform management are only configuration software, bare software, or virtualiser software.

Examples of Virtualiser software are hypervisors and operating systems. Examples of bare software include specific software such as DPAA controllers. As for hardware components, a platform software is said active if it can initiate transactions to hardware elements. Otherwise, it is considered as reactive. For instance, an hypervisor is an active component: it has is own data, and it can issue events through the architecture (e.g., events for scheduling the compute partitions). By the same reasoning, the DPAA software controller is an active component. It can issue IO traffic through the interconnect to the memory. Conversely, a non intrusive trace recorder which snoops the activity of the platform is considered to be a reactive software (provided that it has been proved as being non intrusive).

From an interference point of view, active software running on a hardware/virtual resources can issue transactions through the architecture. As such, they can provoke interferences.

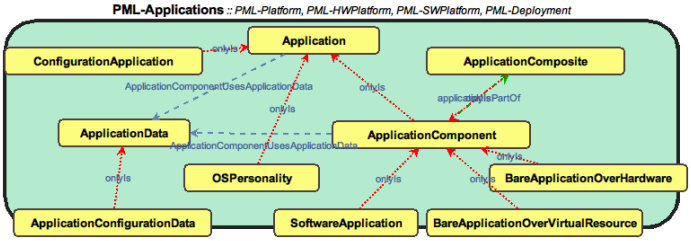

Fig. 4. Meta-model of the application chapter

3) Application chapter: Figure 4 presents the concepts involved in the application chapter. Similarly to the hardware

chapter, the application chapter is a tree structure. It is composed of application components, which are either terminal elements, or elements which themselves break down into sub-applications. The objective is to allow description of complex applications composed of several threads running in several partitions or on several cores. An application that contains sub-applications is called an *Application Composite*.

In this hierarchy, we distinguish three types of terminal application software:

- *Software Application*, whose execution is supported by an operating system.
- *Bare Application over Virtual Resource*, which runs in bare mode (i.e., without the support of any operating system) in a virtual resource (for instance a compute partition managed by an hypervisor).
- *Bare Application over Hardware* resource, which runs in bare mode directly on hardware resources.

Note that the OS personality component refers to OS developed (or provided) by application providers. Such OS personalities are not provided at platform level. OS provided by the platform are described in the software platform chapter as *Platform Bare Component* or *Software Virtualiser Component*.

4) Deployment chapter: The last chapter addresses the configuration of the whole equipment. It describes the *Virtual Resources* created and managed by the hardware/software virtualiser components. It also details on which resources (including virtual resources) are hosted the software components and the application components. And finally, it describes the *transaction paths* followed by transactions issued by active components through the architecture. A *transaction domain* is a directed acyclic graph of *transaction paths* from active to reactive and passive components.

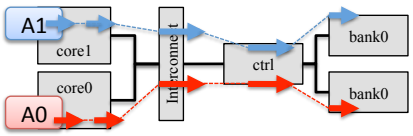

Fig. 5. Toy example

5) A toy example: In order to illustrate the concepts introduced above, let us consider the toy example shown in figure 5. The HW platform involved in this example is composed of two cores, an interconnect, a memory controller, and two memory banks. All these elements are considered to be terminal. The cores, the interconnect, and the controller are reactive, whereas the memory banks are passive components. Two software applications (A0 and A1) run on the platform. Each one is hosted by one core and one bank (cores and banks are not shared between the two applications). We suppose that A0 and A1 run in bare mode (there is no platform software). Finally, we suppose that each application accesses the memory through a dedicated transaction way (one per application). Each way is composed of four transactions paths (one for each traversed component, including the applications). The transaction way for A0 (resp. A1) is depicted in red (resp. blue) in figure 5.

The PML model of this example is described in listing 1.

Listing 1. Toy example (in PML)

```
ToyExampleModel:
  HWPlatform
    /* Hardware components */
    core0, core1, interconnect, ctrl: ReactiveHardware;
    bank0, bank1: PassiveHardware;
    /* Physical connections */
    (core0, interconnect) physicallyConnected;
    (core1, interconnect) physicallyConnected;
    (interconnect, ctrl) physicallyConnected;
    (ctrl, bank0) physicallyConnected;
    (ctrl, bank1) physicallyConnected;

  SWPlatform /* None */

  Applications
    A0, A1: BareApplicationOverHardware;

  Deployment
    /* Software allocation */
    A0 hostedBy core0;
    A1 hostedBy core1;
    /* Transaction description */
    A0_pth, A1_pth, core0_pth, core1_pth, int0_pth,
        int1_pth, ctrl0_pth, ctrl1_pth,
        bank0_pth, bank1_pth: TransactionPath;

    A0_pth hostedBy A0
    A1_pth hostedBy A1
    core0_pth hostedBy core0
    core1_pth hostedBy core1
    int0_pth hostedBy interconnect
    int1_pth hostedBy interconnect
    ctrl0_pth hostedBy ctrl
    ctrl1_pth hostedBy ctrl
    bank0_pth hostedBy bank0
    bank1_pth hostedBy bank1

    (A0_pth, core0_pth) logicallyConnected;
    (core0_pth, int0_pth) logicallyConnected;
    (int0_pth, ctrl0_pth) logicallyConnected;
    (ctrl0_pth, bank0_pth) logicallyConnected;
    (A1_pth, core1_pth) logicallyConnected;
    (core1_pth, int1_pth) logicallyConnected;
    (int1_pth, ctrl1_pth) logicallyConnected;
    (ctrl1_pth, bank1_pth) logicallyConnected;
```

B. Formalisation

Querying and reasoning about the information imply that this information must be formally modeled. To support this approach, we relied on WEIRD, a specification language for expert modeler [BBD+16]. WEIRD provides basic constructions to define concepts, entities, relations between model elements (with the special case of applications), and functions (between model elements and primary types as integer or boolean), as well as a rather rich expression language to describe some model elements and to query the model for assessments.

WEIRD also offers a modularity mechanism: WEIRD specifications can be separated into related *worlds*. When focusing on meta-modeling, this modularity will be used to separate the different concerns of a domain specific modeling language. We will use *world*s to differentiate each PML chapter.

The most interesting aspect of WEIRD with respect to our concern is that the WEIRD semantics are expressed in terms of propositional logic. They allow the specification of formal

queries as first order formulae, and mathematical reasoning over both the queries and the model.

In order to illustrate the WEIRD implementation of the PML chapters, let us first consider the hardware platform chapter. The implementation of this chapter is a WEIRD *world* called "PML-HWPlatform" (partially) shown in listing 2. Each PML concept is implemented as a WEIRD "concept" (a WEIRD concept being equivalent to a type with a sub-typing relation denoted "<:"). Each relation in the PML meta model is then directly implemented as a relation between the corresponding concepts in the WEIRD *world*.

Listing 2. HW platform meta model in WEIRD (partial)

```
world PML-HWPlatform {
  concept HardwarePlatformComponent
  relation physicallyConnected: HardwarePlatformComponent
       x HardwarePlatformComponent

  concept HardwareComposite <: HardwarePlatformComponent
  relation isPartOf: HardwarePlatformComponent
       x HardwareComposite is a container

  concept PassiveHardware <: HardwarePlatformComponent
  concept ReactiveHardware <: HardwarePlatformComponent
  concept ActiveHardware <: HardwarePlatformComponent
  concept HardwareVirtualiser <: ActiveHardware
}
```

The translation of the toy example in listing 1 is then straightforward (cf. listing 3). It is described as a WEIRD *world*. It inherits from the PML chapters (relation *derives* in listing 3), and introduces the concrete elements of the toy example as new *entities* (e.g., core0, core1 and interconnect are three instances of the *ReactiveHardware* concepts).

Listing 3. Toy example in WEIRD (extract)

```
world Toy-Example derives PML-HWPlatform, PML-SWPlatform,
      PML-Applications, PML-Integration {

  entity core0, core1, interconnect, ctrl: ReactiveHardware
  entity bank0, bank1: PassiveHardware

  known (core0, interconnect) in physicallyConnected
  known (core1, interconnect) in physicallyConnected
  ...

  entity A0, A1: BareApplicationOverHardware
  known (A0, core0) in hostedBy
  known (A1, core1) in hostedBy

  entity A0_pth, A1_pth, core0_pth, core1_pth, int0_pth,
     int1_pth, ctrl0_pth, ctrl1_pth,
     bank0_pth, bank1_pth: TransactionPath

  known transactionPathHostedBy(A0_pth)=core0
  known transactionPathHostedBy(A1_pth)=core1
  ...

  known (A0_pth, core0_pth) in logicallyConnected
  known (core0_pth, int0_pth) in logicallyConnected
  ...
}
```

C. Certification requirements revisited (again)

As said previously, one of the benefits of using a formal logical framework is to allow formal verification of certification requirements. In order to illustrate this ability, let us consider two examples of consistency requirements:

- CR1: "*there is no empty composite component*"
- CR2: "*any hardware composite component is not part of itself*"

CR1 and CR2 are quantified requirements. They can be formalized as WEIRD assertions, as shown in listing 4.

Listing 4. Toy example in WEIRD

```
assert CR11 = forall entity  h:HardwareComposite
              | exists entity r::HardwarePlatformComponent
              | (r,h) in isPartOf
assert CR12 = forall entity  a:ApplicationComposite
              | exists entity b::Application
              | (b,a) in isPartOf

assert CR2 = forall entity  h:HardwareComposite
             | not (h,h) in ^isPartOf
```

CR1 is decomposed in two sub-requirements, one related to hardware composite components, and the second one related to application composite components. CR11 assumes that for any hardware composite component h, there exists at least one component r (terminal or not) related to h with the relation *isPartOf*, meaning that r is a sub-component of h.

CR2 is more interesting. Indeed, it uses the transitive closure of the relation *isPartOf* (noted *^isPartOf*). CR2 means that for any hardware composite component, (h, h) is not in the transitive closure of *isPartOf*. In other words, any h is not sub-component of itself nor of any sub-component of itself.

WEIRD being a logical language, such requirements are checked with an automatic solver.

V. INTERFERENCE PREDICTION

Interference prediction is a second key issue when dealing with MCP certification. Let us call "potential interferences" the interferences predicted by the prediction activity, and "actual interferences" the interferences that actually occur in the platform. For a given MCP, let us call I_P the set of "potential interferences" and I_A the set of "actual interferences". The prediction activity must be:

- correct, meaning that $I_A \subset I_P$ (assuming that the platform model is correct, at least all the actual interferences are predicted),
- as accurate as possible, i.e., the number of potential interferences which are not actual interferences must be as small as possible.

A way to answer the correctness issue is to identify a tight (in order to answer the accuracy requirement) sufficient condition for deciding if a configuration may lead to an interference.

A. What is interference?

As defined in [JMR+16], an interference is the disruption of the behavior of an active element A due to the activity of active elements $B_1 \dots B_n$. This disruption is caused by an unexpected behavior of a resource R used by A. The unexpected behavior can be a change of the state of R (e.g., the resource was available and it becomes unavailable, or its value is changed, etc.). The effect of the disruption can be a delay in the activity of A, or a modification of its outputs.

Let us make three remarks:

1) First, note that interferences are not necessarily related to software applications. Any active element can suffer from interferences.
2) Second, an interference involves $n+1$ active components, and at least one resource: the victim of the interference (A), the n components which have caused the interference ($B_1 \dots B_n$), and the resource where the interference occurs (R).
3) Third, this definition is not limited to "*simultaneous collision*" between transactions. Indeed, interferences can occur in scenarios without "*collisions*". For instance when an application B modifies the content of a shared cache by loading its own data and removing data from another application A. When, later, A will try to read its data, it will have to load the data from the memory, that will lengthen its execution time. In such cases, we talk about "*delayed collision*".

Following these remarks, let us propose the following condition:

***Interference condition**: let A and B be two active components, let t_A and t_B two transactions issued by A and B respectively. Let a_1, $a_2 \dots$, a_n the resources crossed by t_A, and let b_1, $b_2 \dots$, b_m the resources crossed by t_B. a_i and b_j can be concrete hardware resources or virtual resources (a memory partition for instance). Then, we state that a potential interference may occur between A and B if $\exists i, j$ such that a_i and b_j are hosted by the same hardware resource.*

Note that this condition does not mention time. It covers interferences due to "*simultaneous collisions*" and interferences due to "*delayed collisions*". One can thus observe that if A can interfere on R with B_1, and A can also interfere with B_2 on R, then A can potentially interfere with both B_1 and B_2 on R. That means that if we are able to predict binomial interferences (i.e., interference involving two active components on one resource), then we can predict any n-nomial interference (i.e., interference involving n active components on one resource) by overlapping binomial interferences.

This shows that it is sufficient to compute binomial interferences by using the interference condition above.

B. Potential interference computation using WEIRD

Having a sufficient condition to infer potential interferences, the next step is to infer them from a PML model.

To achieve this, we use the facts system of the WEIRD logical framework. A "fact" is an expression of the form: *for (typed) variables such that the "selection expression" holds, add "knowledge" to the model*. A fact system can be used to encode rules that automatically complete a model. A simple example is a fact encoding that a binary relation r is symmetric by adding the symmetrical case every time a couple is inserted in r by the user. For instance, to encode that the relation *physicallyConnected* of the PML hardware platform meta-model is symmetric, we add the following fact to the WEIRD implementation:

fact any (e_1, e_2) *in physicallyConnected* | *true* $\Longrightarrow$ (e_2, e_1) *in physicallyConnected*.

Similarly, we encode the interference condition with the two WEIRD facts shown in listing 5. The first fact infers knowledge used by the second fact. It says:

- for any two different transaction paths t_{a_1} and t_{b_1},
- if t_{a_1} and t_{b_1} are active (i.e., issued by active components),
- and if there exist two transaction paths t_a and t_b respectively reachable from t_{a_1} and t_{b_1} (reachability is encoded by the transitive closure of *^logicallyConnected*),
- and if t_a and t_b are collocated (i.e., hosted by the same component),
- then WEIRD infers the following new knowledge: $(t_{a_1}, t_a, t_b, t_{b_1})$ is a potential binomial interference, which can potentially occur on the resource shared by t_a and t_b.

Listing 5. Interference condition in WEIRD (partial)

```
// Fact Interference1
fact any entities
   ta1::TransactionPath,
   ta::TransactionPath,
   tb::TransactionPath,
   tb1::TransactionPath
   | (IsTransactionPathActive(ta1)
     and IsTransactionPathActive(tb1)
     and ta1!=tb1
     and ((ta,tb) in TransactionPathCollocated))
     and ((ta1,ta) in ^logicallyConnected)
     and ((tb1,tb) in ^logicallyConnected)
     ==> (ta1,ta,tb,tb1) in transactionInterference

// Fact Interference2
fact any entities
   ta1::TransactionPath,
   ta::TransactionPath,
   tb::TransactionPath,
   tb1::TransactionPath
   | ((ta1,ta,tb,tb1) in transactionInterference
      ==> (transactionPathHostedBy(ta1),
           transactionPathHostedBy(ta),
           transactionPathHostedBy(tb1))
               in resourceInterference
```

Then, the second fact encodes the interference predicate at resource level: if $(t_{a_1}, t_a, t_b, t_{b_1})$ is a potential binomial interference between transactions, then (a_1, r, a_2) is the resulting potential interference at resource level, where a_1 (resp. a_2) is the resource which hosts the path t_{a_1} (resp. t_{b1}), and r is the shared resource which hosts both t_a and t_b.

C. Application to the toy example

Let us consider the toy example once more. Applied on this example, the previous facts produce the new knowledge:

```
[Facts] -- Fact Interference1 added knowledge:
   ((A0_pth, int0_pth, int1_pth, A1_pth)
     in transactionInterference)
   ((A0_pth, ctrl0_pth, ctrl1_pth, A1_pth)
     in transactionInterference)
[Facts] -- Fact Interference2 added knowledge:
   ((A0, interconnect, A1) in resourceInterference)
   ((A0, ctrl, A1) in resourceInterference)
```

As expected, two interferences are predicted between $A0$ and $A1$: the first one on the interconnect, and the second one on the controller.

VI. Case-study

As a more complex case-study than the toy example discussed in the previous section, let us consider the many-core processor Kalray MPPA®-256 [Kal12] (see also [Per17] for a clear and detailed presentation of the Kalray MPPA®-256 architecture).

A. Kalray MPPA®-256

The Kalray MPPA®-256 is a many-core processor featuring 288 cores on a single chip. It is organized in 16 compute clusters and 4 I/O clusters serving as interfaces for managing communications with out-of-chip components and hosting the global memory of the processor. All the clusters are connected on a dual NoC (Network on Chip) enabling point-to-point communications (one NoC for data communication (D-NoC), and a second one for control messages (C-NoC)). In this case-study, we only consider one compute cluster connected to the D-NoC. A compute cluster features: (1) 16 cores, (2) one additional core denoted as Resource Manager (RM) in charge of managing the cluster's local resources, (3) one DMA unit to manage data transfer from and to the dual NoC, (4) a shared memory organised in 16 independent banks, (5) a full-duplex access point to the NoC (NoC Router), and (6) a Debut and Support Unit (DSU). The DMA unit is divided in two parts: a sending part (DMATx) and a receiving part (DMARx). Similarly, the NoC Router is composed in two parts: the first one connected to the D-NoC, and the second one connected to the C-NoC. Each part is composed of two FIFOs (one output FIFO to the D-NoC, and one input FIFO from the NoC).

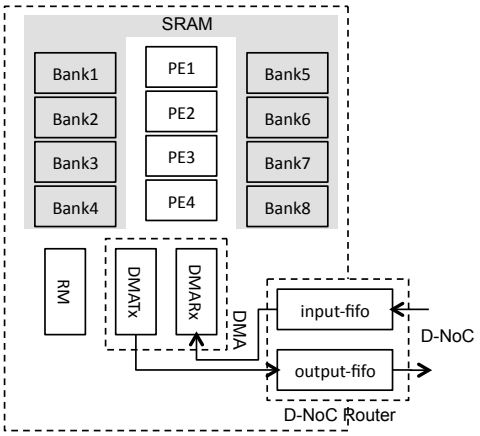

Fig. 6. Architecture of compute clusters in the Kalray MPPA®-256 (restricted to 4 PEs and 8 Banks)

In this case-study, we only consider a restricted version of the MPPA compute cluster composed of 4 PEs, 8 Banks, 1 RM, 1 DMA and 1 D-NoC Router as depicted in Figure 6. As such, a compute cluster can be seen as a quad-core processor. However, conversely to classic multi-core processors, transactions between HW elements (PEs, DMA and RM) and the shared memory are supported by a hierarchical arbitrer, as depicted in Figure 7. For each Bank_i ($i = 1 \dots 8$), transactions to the bank are interleaved as follows:

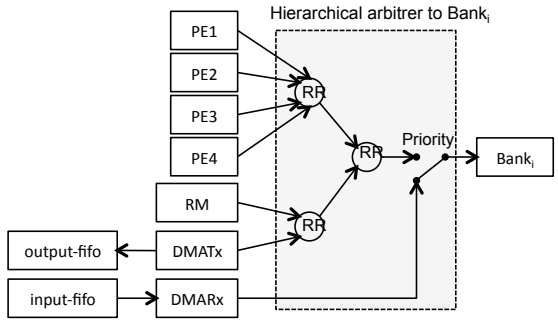

Fig. 7. Hierarchical SRAM arbitrer of Kalray MPPA®-256

- PEs accesses are arbitrated in a Round-Robin fashion.
- RM and DMATx accesses are arbitrated in Round-Robin.
- The two resulting transactions (from PEs and from RM or DMATx) are then arbitrated in Round-Robin.
- Finaly, the resulting transactions from this last Round-Robin arbitrer is interleaved with memory requests from DMARx. However, contrary to the previous arbitration policies, requests from DMARx are treated with full priority over other memory accesses.

As a consequence, requests from PEs, RM or DMATx are systematically stalled while the DMA writes input data to the memory. This is done to avoid congestion at the entry point of the cluster.

From a global point a view, the scheme depicted in Figure 7 is replicated for each bank as shown in Figure 8. That means that two different masters (among PEs, the RM and the two DMA parts) can access two different banks in parallel, without suffering from interferences.

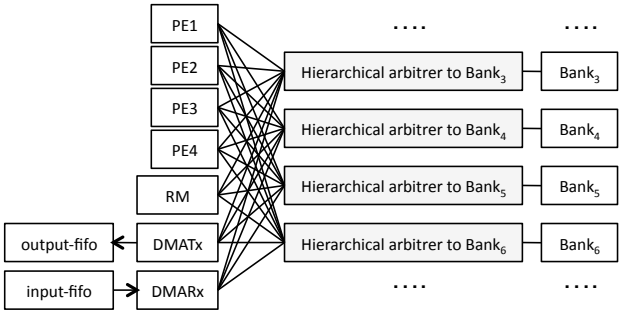

Fig. 8. Intra cluster communication architecture

Note that, in addition to communication through shared memory, the master elements of a compute cluster (PEs, RM, DMA) can also interact by sending events:

- PEs and RM can send events to each other to achieve a synchronisation barrier;
- PEs can send events to DMATx and DMARx in order to configure the Tx and Rx channels.

B. Modeling

To model a Kalray MPPA®-256 compute cluster, we follow the same approach previously shown for the toy example

in section IV-A5. Each HW component (PE, RM, DMATx, DMARx, FIFO, Bank, Arbitrer, etc.) becomes a WEIRD entity. The full WEIRD model contains 51 hardware components, 90 physical connections between hardware components, 196 transaction paths, and 206 logical connections between transaction paths[1].

C. Interference prediction

We have studied two compute cluster configurations: a "full" configuration, and a "final configuration.

a) "Full" configuration: In the first experiment, we consider a "full" configuration, i.e., a configuration in which all transactions are allowed between the masters elements (PEs, RM, and DMA) and the shared memory. Event sending between PEs, the RM and the DMA is also fully allowed. The interference prediction method computes a set of 876 potential interferences. Let us call I_{Full} this set. The computation of I_{Full} took about 37 minutes using about 6Go memory on a 2.3 Ghz Core i7 processor with 16Go memory.

Interferences of I_{Full} are mainly due to: (1) collisions inside each hierarchical arbitrer between transactions from PEs, the RM, and the DMATx, (2) collisions inside each master element between the own activity of the master element and events sent by other master elements, and (3) collisions inside each bank of the shared memory. As expected, input transactions from the D-NoC do not suffer from any interference when crossing the hierarchical arbitrers to the banks. The prediction method takes into account the fact that packets from the DMARx have full priority over other packets (from PEs, RM, DMATx). However, I_{Full} contains 44 interferences in which input transactions are potentially victim of disruptions due to other transactions. These interferences occur:

- at the entry of each bank; despite the fact that DMARx requests have a higher priority than other request, they have to wait before entering the bank if the memory is serving another request arrived just before it (from PEs, the RM or DMATx); in that case, the DMARx requests could be delayed by at most by one memory transaction from another master.
- in the DMARx component, each time a PE sends an event to DMARx (for instance to access the Rx registers).

b) "Final" configuration: In the second experiment, we consider a "final" configuration, representative of an embedded application, in which:

- each PE is associated with two private banks (PE1 with bank1 and bank2, PE2 with bank3 and bank4, etc.) as suggested in [Per17] (configuration achieved by appropriate MMU settings);
- only the RM is allowed to send events to the PEs;
- DMARx is only allowed to write input data in (and then to access) bank1, bank3, bank5 and bank7;
- and DMATx is only allowed to read output data from (and then to access) bank2, bank4, bank6 and bank8.

[1]The full model is available at: http://w3.onera.fr/phylog

The interference prediction method returns a set I_{Final} of 51 potential interferences. The computation of I_{Final} took about 35 minutes using about 6Go memory.

As expected, $I_{Final} \subset I_{Full}$. I_{Final} is the set of the remaining interferences not avoided by the "final" configuration and that must be addressed by the certification argumentation. As expected in this configuration, DMARx no longer suffers from disruptions due to events sent by PEs. However, input transactions from the D-NoC still suffer from 4 interferences $(\textit{input-fifo}, \textit{bank}_i, \textit{PE}_i) \in I_{Final}\ i = 1 \ldots 4$ (one interference at the entry of each bank with the PE "owner" of the bank).

VII. DISCUSSION AND CONCLUSION

The aim of the approach presented above is to automatically generate of the list of interferences that can potentially occur in a multi-core equipment. According to the MCP-CRI, each one of those interferences is a key certification issue. It has to be addressed by a dedicated argumentation showing that the interference never happens or has a negligible effect on the behavior of the applications hosted by the equipment. For instance, let us consider again the Kalray MPPA®-256 case-study developed in the previous section. The set I_{Final} returned by the prediction interference analysis contains 51 potential interferences. The certification files should thus contain 51 argumentation trees, one for each potential interference.

However, the approach still suffer from several limits.

Combinatorial explosion. The first limit to overcome is the combinatorial explosion encountered when computing the intersection of the transaction ways. For instance, to consider a real Kalray MPPA®-256 cluster composed of 16 PEs and 16 banks. This would exceed the capacity of the WEIRD tool suite. To overcome this limitation, future work will include the study of a more efficient implementation of the interference prediction analysis.

Taking into account the real time behavior of the transactions. A second perspective for future work is to consider the temporal behavior of the transactions. For example, let us consider again the "final" configuration of the Kalray MPPA®-256 case-study. In this configuration, the input transactions from the D-NoC suffer from interferences at the entry of the banks. Let us now suppose that DMARx is temporally interleaved with each PE_i on bank_i (as defined by the time triggered execution model proposed in [Per17]). By configuration, PE_i and DMARx can both access bank_i during non overlapping periodic time intervals. As a result, the 4 interferences (between input-fifo and PE_i on bank_i) actually never happens. Taking into account the real-time behavior of the transactions thus leads to a more accurate interference prediction.

Taking into account the faulty behavior of the transactions. The third perspective for future work on the approach presented in this article is thus to pay attention to the effect of component failures on the identified interferences. To do so, we will take inspiration from the Model-Based Safety Assessment (MBSA) approach [BP17].

REFERENCES

[AAAC17] 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 (SIES17)*, 2017.

[BBD+16] Pierre Bieber, Frdric Boniol, Guy Durrieu, Olivier Poitou, Thomas Polacsek, Virginie Wiels, and Ghilaine Martinez. MIMOSA: Towards a model driven certification process. In *8th European Congress Embedded Real Time Software and Systems (ERTS216)*, 2016.

[BP17] Marco Bozzano and Yiannis Papadopoulos, editors. *Model-Based Safety and Assessment - 5th International Symposium, IMBSA 2017*, volume 10437 of *Lecture Notes in Computer Science*. Springer, 2017.

[CAS14] CAST (Certification Authorities Software Team). Position Paper on Multi-core Processors - CAST-32, 2014. Retrieved from https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/media/cast-32.pdf.

[EAS16] EASA (European Aviation Safety Agency). The Use of Multi-Core Processors in Safety-Critical Applications - CRI, 2016.

[Hol15] C. Michael Holloway. Explicate '78: Uncovering the implicit assurance case in do-178c. In *23rd Safety-Critical Systems Club (SCSC) Annual Symposium*, February 2015.

[JGBF12] 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.

[JMR+16] Xavier Jean, Laurence Mutuel, Didier Regis, Hlne 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.

[Kal12] Kalray Corporation. *The MPPA hardware architecture*, 2012.

[KW04] Tim Kelly and Rob Weaver. The Goal Structuring Notation A Safety Argument Notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[MoD03] MoD. *Defence Standard 00-42 Issue 2, Reliability and Maintainability (R&M) Assurance Guidance Part 3 R&M Case*, 2003.

[MoD07] MoD. *Defence Standard 00-56 Issue 4, Safety Management Requirements for Defence Systems Part 1 Requirements*, 2007.

[Per17] Quentin Perret. *Predictable Execution on Many-Core Processors*. PhD thesis, Universit de Toulouse, ISAE, 2017.

[Pol16] Thomas Polacsek. Validation, accreditation or certification: a new kind of diagram to provide confidence. In *IEEE Tenth International Conference on Research Challenges in Information Science (RCIS'16)*, 2016.

[Tou03] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 2003. Updated Edition, first published in 1958.

[WEE+08] 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.

[WR12] 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.