Towards the Certification of Hybrid Architectures: Analysing Interference on Hardware Accelerators through PML

Benjamin Lesage, Frédéric Boniol, Kevin Delmas, Adrien Gauffriau, Alfonso Mascarenas Gonzalez, Claire Pagetti

Towards the Certification of Hybrid Architectures: Analysing Interference on Hardware Accelerators through PML

Benjamin Lesage[1], Frederic Boniol[1], Kevin Delmas[1], Adrien Gauffriau[2], Alfonso Mascarenas-Gonzalez[1], Claire Pagetti[1]
[1] ONERA, Toulouse, France, [2] Airbus, Toulouse, France

***Abstract*—The emergence of Deep Neural Network (DNN) and machine learning-based applications paved the way for a new generation of hybrid hardware platforms. Hybrid platforms embed several cores and accelerators in a small package. However, in order to satisfy the Size, Weight and Power (SWaP) constraints, limited and shared resources are integrated. This paper presents an overview of the standards applicable to the certification of hybrid platforms and an early mapping of their objectives to said platforms. In particular, we consider how the classification of AMC20-152A for airborne electronic hardware applies to hybrid platforms. We also consider AMC20-193 for multi-core platforms, and how this standard fits different types of accelerators.**

I. Introduction

New software paradigms and capabilities drive the demand for additional computing power in avionic systems. Hybrid architectures can, in a small SWaP package, support this demand. They embed on the same platform general-purpose cores, and specialised accelerators which can support some of the additional workload. However, like any other hardware platform, they need to go through a stringent certification process before they are deployed in avionic system.

The European Union Aviation Safety Agency (EASA) and Federal Aviation Administration (FAA) respectively define Acceptable Means of Compliance (AMC) and Advisory Circulars (AC), setting down objectives applicants to the certification process satisfy. The joint A(M)C AMC20-152A and AMC20-193 in particular define objectives for the respective certification of hardware platforms and multi-core processors.

The PHYLOG methodology [1] was proposed as mean of supporting applicants, especially regarding AMC20-193 on multi-core processors. PHYLOG is based on the definition of argumentation patterns for the certification objectives in AMC20-193, with each objective decomposed in supporting claims, strategies, or evidences. At the core of the methodology, the PHYLOG Modelling Language (PML) [2] captures knowledge about a platform, both hardware and software aspects, and their configuration. PML supports analyses to fulfil claims in the certification patterns instantiated for the platform.

The contributions of this paper are to present an overview of the objectives applicable to hybrid platforms. We also identify the issues related to modelling the accelerators in such platforms and propose related PML model templates. This paper is organised as follows. Section II briefly recaps the PHYLOG methodology, with Section III providing an introduction to PML. An example of accelerator and its hybrid platform is introduced in Section IV to support further discussions and examples. In the context of hybrid platforms, we identified two relevant AMC: AMC20-152A [3] and AMC20-193 [4] discussed respectively in Section V and Section VI. Section VII briefly discusses related work, before Section VIII recaps the discussion and outlines perspectives.

II. PHYLOG Methodology

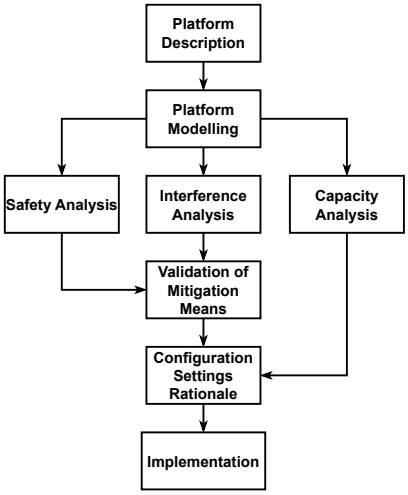

Fig. 1. Overview of PHYLOG methodology

The PHYLOG methodology [1] describes the activities to produce the elements for instantiating the PHYLOG argumentation patterns. These patterns were derived from the objectives defined in AMC20-193, on multi-core processors, to build an argumentation strategy for certification. They decompose the top-level AMC objectives into supporting claims, strategies, evidence, and warrants. An overview of the methodology is presented in Figure 1. It is composed of eight main activities:

- **Platform description** captures the knowledge about the platform characteristics based on the available documents and the applicant's assessments. It also captures the target configuration, including hardware and software settings such as the mapping of applications hosted on the platform to cores.

- **Platform modelling** formalises the platform description knowledge in order to support further analyses. It is based on PML. While not an objective of AMC20-193, it allows running the supporting automatic safety, capacity and interference analyses in order to contribute to said objectives.
- **Safety analysis** identifies and evaluates the failures and alterations which can affect the platform and hosted applications.
- **Interference analysis** enables the identification of interferences via interference calculus and the classification of their effects.
- **Capacity analysis** enables the verification of shared resources' usage, ensuring the demand for resources of the platform never exceeds their capacity.
- **Validation of mitigation means** encompasses the design and validation of mitigation means for failure, interference, and other alterations identified in earlier activities.
- **Configuration settings rationale** justifies that all configuration settings support the requirements on the platform, or are harmless to them.
- **Implementation** concerns the certification of the system implementation on the platform. It is associated with the DO-178C standard and out of the context of PHYLOG.

Note that the activities form an inherently iterative process. As an example, the interference analysis may highlight a misunderstood interference channel, feeding back into the platform description and its model.

We focus in the following on the platform aspects (description and modelling), as they are the most relevant to hybrid platforms. We consider specifically the use of PML, and its limitations, to model accelerators. The use of PML would thus allow for the application of existing PHYLOG-based analyses [1], discussed in other work for interference or safety, to instantiate the PHYLOG certification patterns for hybrid platforms. PML is introduced in the next section.

III. PML

PML, the PHYLOG Modelling Language [2], is a Domain Specific Language embedded with the SCALA language to capture the description of a platform. A hardware platform is modelled in PML as a collection of components, capturing the functional blocks of a multi-core processor, e.g. a core, cache, memory, or bus, and links between components. Composite components encapsulate one or more components, composite or atomic, to allow for the hierarchical specification of a model. *Atomic* components provide generic services to the software hosted by the platform, such as a *load* from the main memory or a *store* to a configuration register.

The relationship between a component and other services of the platform defines its role in the model. *Initiator* components, such as a core, call services from other components on the platform, most often as a result of software running on the initiator, be it a user application or platform-embedded micro-code. *Target* components, such as the main memory, expose services to satisfy transactions from other components. *Transporter* components, such as an interconnect, process transactions between an initiator and its target.

A *transaction* is a footprint of a use of the platform by a software component. A transaction more formally captures the set of components, and their services, used by a request from an initiator to a target. A transaction must follow a valid path in the platform, through the links between its components. Services thus model the dependencies between the software and the hardware.

Example 1. *To exemplify the use of PML, we consider a representation of the* KEYSTONE *TCI6630K2L from Texas Instruments. An overview of the* KEYSTONE *is presented in Figure 2. It is composed of a four C66 DSP pack where cores are characterised by dedicated L1 and L2 caches, and a memory extension and protection unit (MPAX). The platform also comprises a 2* ARM *A15 pack where cores are characterised by dedicated L1 caches, memory management units (MMU), and a shared L2. In addition, it includes a central memory system giving access to SRAM and external DDR. Memory accesses are managed by a Multicore Shared Memory Controller (MSMC). A set of I/O and utility peripherals (e.g. GPIO, UART, boot) is also present on the platform and an ultra speed bus (TeraNet) connects the peripherals, the memories, and the cores altogether.*

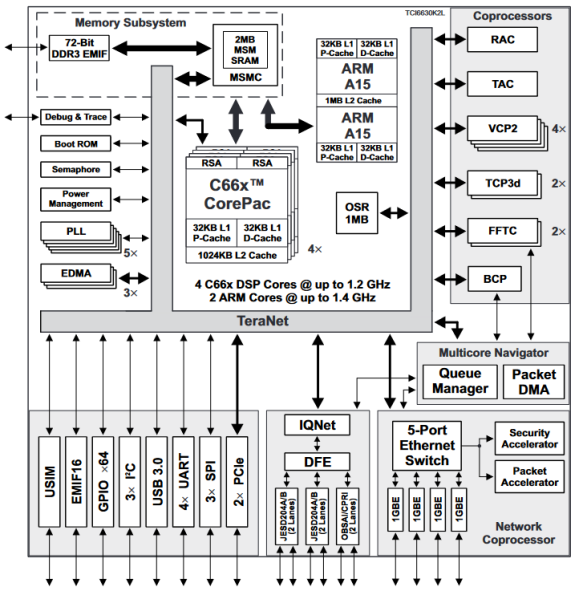

Fig. 2. Overview of the TI KEYSTONE TCI6630K2L

Figure 3 illustrates a PML model for a simplified version of the KEYSTONE[1]. *This basic model includes:*

- *Cores as initiators: 4 C66 DSP, and 2* ARM *A15 cores;*
- *Memories as targets: DDR, SRAM, and all caches;*
- *Peripherals as targets: GPIO, I2C, SPI port, PCIe, etc.;*

[1]For the sake of brevity, coprocessors have been omitted, as well as implicit links between stacked components. Peripherals have been simply classified as targets.

- *Buses and Memory protection units as transporters: the TeraNet bus connected to the Memory Shared Multicore Controller (MSMC), memory and cache controllers, etc.*

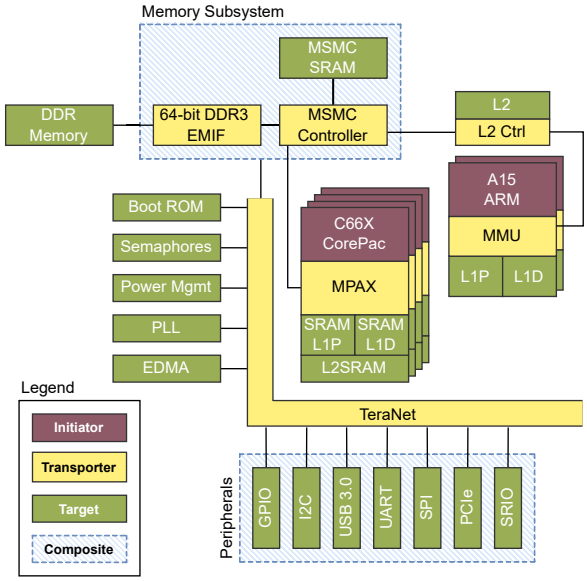

Fig. 3. Simplified PML model for the KEYSTONE platform

IV. HYBRID ARCHITECTURES - THE GPU EXAMPLE

To support the discussion around hybrid platforms, we introduce an example of accelerator: Graphical Processing Units (GPU). Compared to traditional CPUs, GPUs feature numerous cores with simpler control flow but efficient data ones. GPU cores tend to work in a lockstep-like fashion called Single Instruction Multiple Threads (SIMT) in reference to SIMD (Single Instruction Multiple Data). Internal scheduling policies on the GPU aim to maximise core occupancy and throughput. With their focus on high-throughput floating point computation, GPU are well suited to the acceleration of neural network workloads. Their reuse has been facilitated by the advent of General Purpose GPU programming frameworks (GPGPU).

There has been considerable effort to characterise the behaviour of GPU accelerators, in particular work on NVIDIA GPU [5], [6], [7], [8] and the assorted GPGPU CUDA software stack [9], [7], [10], [8]. These efforts highlight the difficulty of characterising complex, multi-core, COTS (Commercially available Off-the-Shelf) platforms. To the best of our knowledge, PasTiS [6] and the hybrid analysis in [11] are some of the few efforts to build a GPU model respectively for static and hybrid WCET analysis. The inherent parallelism at the application-level, as opposed to instruction-level like vectorised arithmetic units [12], [13], can also pose problems for WCET and interference analyses [14].

Example 2. *The NVIDIA Jetson AGX Xavier [15] is a high-performance SoC designed for embedded systems. The Xavier uses an 8-core "Carmel"* ARM *processor, organised in clusters of 2 cores. The "Carmel" processor complies the* ARM *v8.2A specification, but it is unclear if it is based off an existing* ARM *design (e.g. the Cortex-A78) and the level of customisation introduced by NVIDIA. The Xavier features amongst other accelerators a GPU using the Volta architecture, highlighted in Figure 4. The GPU is composed of 512 cores, grouped in 8 Streaming Multiprocessors (SM). The Volta GPU shares a memory fabric with other accelerators, and the memory controller with the CPU.*

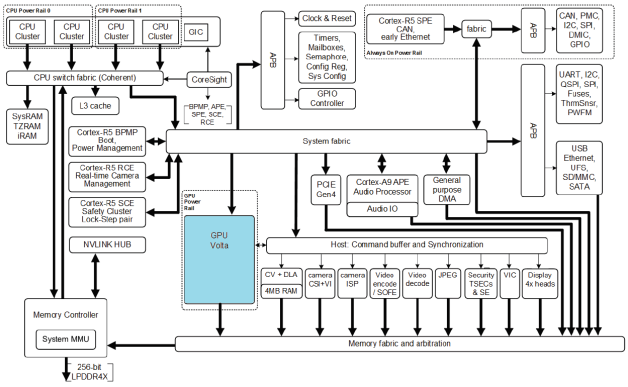

Fig. 4. Overview of the NVIDIA Jetson AGX Xavier

We present in Figure 5 a high-level PML *model of the NVIDIA AGX Xavier SoC. Fabrics and backbones act as transporters for the components of the system. The main memory is a target shared by the CPU and the GPU. The cores of the "Carmel"* ARM *processor act as multiple initiators. As for the* KEYSTONE*, we currently omit coprocessors and peripherals from the classification. A key question is:* How to model a complex accelerator like the Volta GPU? *It acts as an initiator, causing interference on the main memory and the controller fabric, and as a target for commands from the CPU.*

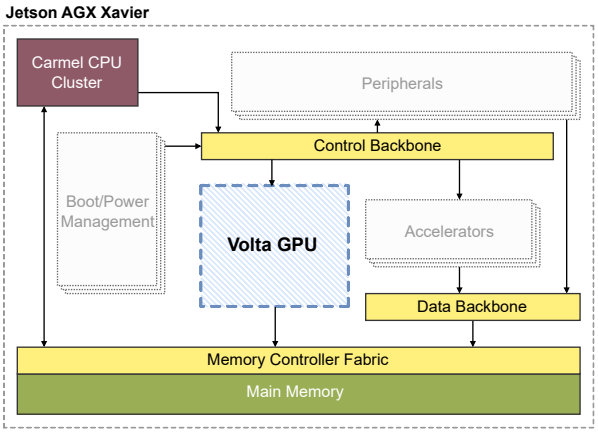

Fig. 5. Simplified PML model for the NVIDIA AGX Xavier

V. AMC20-152A ON HYBRID ARCHITECTURES

AMC20-152A discusses the certification of existing (COTS) or newly-developed platforms, the distinction between the two, and the objectives relevant to each.

A. Overview of the AMC20-152A

The ED-80/DO-254, both dated from the year 2000, define guidance for the design of airborne electronic hardware. The AMC20-152A aims to provide additional guidance and clarification. It is thus complementary to the AMC20-193 on multi-core platforms. The clarifications proposed by the AMC20-152A are important, as devices, especially COTS, become more complex and integrate in a single chip more functions than older ones. The AMC20-152A objectives are classified according to whether they apply to complex custom devices, COTS IP (design functions used to design and implement a custom device, be it a PLD, a FPGA or an ASIC), or COTS devices[2]. Applicants for certification must address them as part of the Plan for Hardware Aspects of Certification (PHAC), or related planning documents.

The first distinction in the AMC20-152A is between COTS and custom functions. COTS functions (IP or devices) are, as the name implies, commercially available, off-the-shelf. The AMC20-152A recognises the risks inherent to the use of COTS, and incomplete or incorrect documentation. COTS may not have been developed within the ED-80/DO-254 standard or avionic applications, nor have sufficient service experience. The development assurance for COTS items (hardware or software components ED-80/DO-254) thus follows different objectives from custom devices. Items developed and fully controlled by the applicant cannot classify as COTS. Those items may however be previously developed hardware, which may take credit from prior deployment and in-service experience provided their new function, usage and environment conditions do not invalidate the original design assurance.

The key objectives of the process for COTS items are 1) identifying used functions, and 2) assessing correct use of the COTS item. The used functions need to support the system requirements on the device. Unused functions, such as unused cores on a MCP (as per AMC20-193), need to be properly deactivated, with means of mitigation to prevent their inadvertent activation. Correct use of a COTS item requires to assess its integration against the operation conditions, such as temperature or input parameter ranges, defined by the manufacturer. This may preclude the use of undefined or undocumented configurations, unless their reliability can be established. The identification of failure modes[3] and the item configuration also need to be considered. This includes identifying if any microcode may contribute to a used function. Microcode is a hardware-level set of instructions, typically stored in the COTS item. It may be qualified by the manufacturer, if left unmodified, or require a separate mean of compliance.

Devices are further classified into simple or complex ones as defined by the ED-80/DO-254. The classification captures whether a comprehensive verification of the device is realistic. It must be explicit, and justified for simple devices (custom or COTS). The simplicity of a device relies on the simplicity and independence of all its functions, interfaces, building blocks, etc. The composition of simple items may therefore be a complex item.

[2]We omit circuit boards assemblies (CBA), as the AMC20-152A in practice redirects to ED-80/DO-254.

[3]Single Event Effects (SEE) are explicitly omitted from the AMC20-152A scope.

B. Considerations for accelerator-related objectives

As per the AMC20-152A, most hybrid or multi-core architectures should fall under the definition of complex devices with multiple processing elements interacting. The Platform description and modelling phase for custom models, including any accelerator, will directly benefit from the AMC20-152A objectives' outcome, notably the conceptual and detail designs, and the device verification. For COTS functions, as prescribed by the AMC20-152A objectives, a PML model should be built from the manufacturer specification supplemented by characterisation and verification activities. COTS IP specifically may provide detailed information on the function based on the stage of the design where they are instantiated, from Hard IP, embedded in the silicon by the manufacturer, to Soft ones, captured by a hardware description language. Microcode, if present on used functions, needs to be considered as part of the platform model, as transactions between components.

We identified 4 activities for hybrid platforms and accelerators, per AMC20-152A objectives: **Activity 1:** An assessment should be performed for each device or its integration, as they may fall under different classifications: COTS, custom, soft IP, hard IP, multi-core processor... In particular, one should consider how the device is configured and accessed through hardware and software means, how it interacts with the rest of the system, and whether or not existing analysis techniques and tools apply.
Activity 2: It is necessary to master *complex* core architectures. More specifically stressing benchmarks would be needed in addition to documentation reviews.
Activity 3: The utilisation of COTS must be within the limit of the device manufacturer specification. This means that we need a specification of the COTS and its limits to check the compliance of usage.
Activity 4: It is mandatory to qualify the COTS behaviour and all micro-code, as defined in AMC20-152A (Section V-A).

VI. AMC20-193 ON HYBRID ARCHITECTURES

The AMC20-193 was extensively studied in PHYLOG to define a certification methodology specifically for multi-core platforms [1]. We provide a brief summary of AMC20-193 in the following.

A. Overview of the AMC20-193

The AMC20-193 defines a Multi-Core Processor (multi-core processor) as a device with two or more activated processing cores, with a core being a device that executes software. The AMC20-193 recognises two exceptions to the definition of active cores, cores in lockstep executing the same software and inputs to compare their output; and cores connected solely through data buses typically used in avionics systems.

The AMC identifies both temporal and functional interference. Interference occur when the behaviour of an application varies over its behaviour in isolation when running in parallel with others. Interference occur as a result of shared hardware or software resources of the multi-core processor. As an example, interference may cause additional delays due to the arbitration of accesses to the resource or control flow variations due to external modifications of a shared variable. Interference may cause a loss of deterministic behaviour for the application.

All software components should exhibit correct functional and timing behaviours in the presence of interference. The AMC thus defines an interference channel as "a platform property that may cause interference between software applications or tasks". The impact of interference channels on applications in the system should be assessed. The planning objectives in AMC20-193 require the identification of shared resources, their use by, and their allocation to software applications, where applicable. This aims to first ensure the overall demand for resources at any given time does not exceed the available resources' capacity, and second to avoid or mitigate interference. Mitigations should be deployed and verified for impactful interference channels. The definition of an interference channel in the PHYLOG methodology is a conservative one, in line with the AMC objectives.

The objectives require all software hosted on the multi-core processor to be identified, including applications, operating systems, hypervisors, as well as libraries and runtime. The AMC20-193 prescribes that any component for which interference is mitigated, possibly at the platform-level through robust partitioning, may be separately analysed and verified. Otherwise, they should be tested on target with all other software components under the final configuration. The PHYLOG methodology, and in particular interference calculus, can help assessing whether a modelled accelerator or a platform supports robust partitioning, by identifying interference channels, their impact, and that of any deployed mitigation (through benchmarking).

The question in the context of accelerators, is whether or not the PML model is suitable to model them, and whether and how it should be extended. Let us now characterize what type of resource is an accelerator. We have identified 3 dimensions to take into account.

B. Dimension 1

The first dimension concerns the number of applications that can simultaneously access the accelerator. We define two categories within that dimension:

- those that can be accessed solely by one application at any given time are called *unitary accelerators*;
- those that can be accessed by multiple applications simultaneously are called *parallel accelerators*.

Note that the classification of an accelerator as *unitary* may be inherent to the accelerator itself, e.g. if it cannot support multiple applications by design, or enforced by the platform, e.g. through application design or partitioning mechanisms.

C. Dimension 2

The second dimension concerns how the accelerators are connected to the core and how the workload is launched. In that dimension, we have identified four categories. The simplest case concerns *tightly coupled* accelerators.

Category 1. ***Tightly coupled accelerator.*** *The accelerator, as an example a vectorised functional unit, operates in the context of a complex core; all transactions effectively originate from the core operations and transit through the core interfaces.*
Modelling impact on **PML.** *The core is still modelled as the sole initiator. Such an accelerator can only be unitary, as a core executes only one application at any given time*[4]*. However transactions caused by an application using the accelerator may present a different profile.*

Example 3 (of category 1)**.** *The* ARM *A15 [16] cores can include a NEON VAU and floating point execution unit. SIMD Load/Store instructions allow for transfers between NEON registers and the memory. Vector accesses target one or more lanes of the same or of consecutive vector registers. The architecture thus does not guarantee the atomicity of the access to the memory even for scalar accesses. Each instruction can generate multiple transactions depending on the access size, the alignment of the address and the memory segment. Served by the private or shared caches, or the main memory, SIMD Load/Store may be subject to high timing variability and interference.*

The A15 cores in the KEYSTONE *presented in Example 1 do feature a NEON VAU. As discussed, the core is still modelled as a single initiator and the model in Figure 3 remains valid even when the NEON is in use.*

The second case concerns *passive* accelerators that are controlled by a remote core, e.g. via configuration registers. A passive accelerator cannot generate any transaction to access any shared resource and is thus a target that can be shared by several cores.

Category 2. ***Passive accelerator.*** *The accelerator is a resource used by the core(s). It behaves from a high level point of view like a DDR that receives requests for* load *and* store*.*
Modelling impact on **PML.** *It can be abstracted as a target. Two or more applications using the accelerator concurrently would be assumed to interfere. Thus it could be unitary or parallel, but in both cases it will be modelled in the same way. The transactions caused by the controlling core may present a different profile.*

Example 4 (of category 2)**.** *The NVIDIA Deep Learning Accelerator (NVDLA) outlined in Figure 6 is an accelerator developed by NVIDIA, with both open-source hardware and software. The NVDLA is a complex COTS device. Tailored to neural network applications, it features functional blocks dedicated to convolution, activation functions, pooling, normalisation, or reshaping operations. The blocks can operate*

[4]AMC20-193 explicitly excludes hyperthreading.

independently, performing memory-to-memory operations, or pipelined, passing data to each other to avoid the memory round-trip. The memory (DBBIF), interrupt (IRQ), and configuration (CSB) interface can be connected to various protocols such as ARM *AXI.*

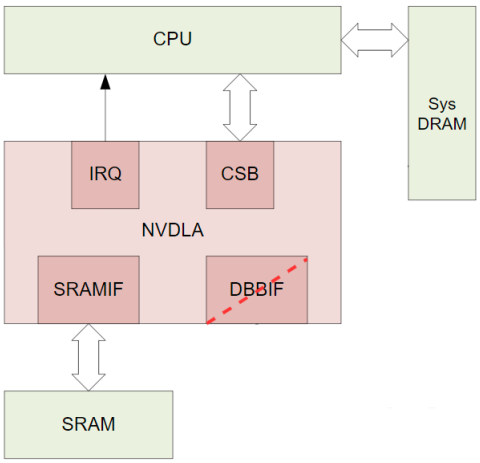

Fig. 6. Integration of the NVIDIA NVDLA in a passive configuration [17]

As a soft IP, the NVDLA exposes all information regarding its internal behaviour which eases the development of a model for timing or interference analysis. The DBBIF, CSB, and target memory subsystem are obviously shared resources between functional blocks. The scope and mitigation of any resulting interference however require more information about the NVDLA integration. The device can be included as part of custom devices or available in future COTS platforms.

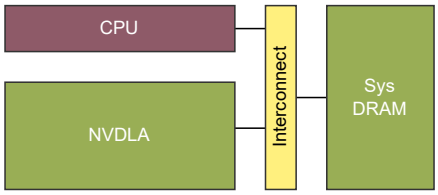

Fig. 7. Simplified PML model for the NVDLA in a passive configuration

Figure 7 presents a PML model for a NVDLA in a passive configuration. The accelerator and all its resources are abstracted as a single target, accessed through the interconnect. Transactions initiated within the NVDLA remain within the device, e.g. from its functional blocks to the CSB or SRAM. As such they would not need to be captured by the model. They are thus implicitly assumed to be non-interfering with external transactions, e.g. from the CPU to the CSB. Such an assumption must be verified during interference analysis.

The third case concerns *semi-active* accelerators. In that situation, the accelerator is triggered by a remote core but it accesses shared resources (e.g. DDR) to *load/store* its data. Thus it generates interferences within the hybrid architecture.

Category 3. ***Semi-active accelerator.*** *The accelerator operates under the control of a core and it behaves from a high level point of view as a DMA that generates requests for* load *and* store *under the impulse of another core. However the precise role of the core needs to be clarified, as well as the interface between the accelerator and the hybrid platform.*
Modelling impact on **PML.** *A unitary* semi-active *accelerator is thus modelled as a single initiator and the profile of the remote core must contain all the transactions needed to configure the accelerator. Parallel accelerators would need more refined analyses to check whether they will be decomposed into one or multiple initiators.*

Example 5 (of category 3)**.** *An example is the NVDLA in a "Small" configuration as depicted in Figure 8. Compared to the passive configuration of Example 4, the NVDLA accesses resources shared with other initiators in the system. The NVDLA [17] could be modelled, as depicted in Figure 9, using a single initiator with interfaces to the system, as no interface or resource between the NVDLA and the controller is shared with other devices. This model assumes a pipelined configuration of execution on the NVDLA, where a single application may use the NVDLA and components do not interfere on the DBBIF. (Example 9 considers a configuration where each functional block is a separate initiator.)*

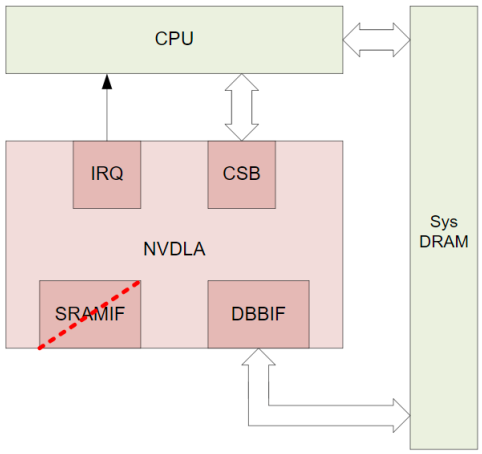

Fig. 8. Integration of the NVIDIA NVDLA in a small configuration [17]

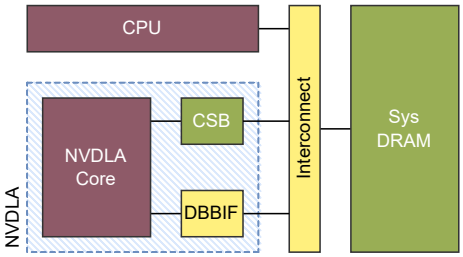

Fig. 9. Simplified PML model for the small NVDLA

Example 6 (of category 3)**.** *The i.MX 8M Plus processor from NXP [18] features, amongst other accelerators, a NPU, e.g. a VIP8000 hard IP from VeriSilicon. The NPU is a complex COTS device. The processor reference manual unfortunately*

provides little information about the NPU, except for the high-level functional description in Figure 10. It probably features VAU and systolic-like blocks as it supports hundreds of multiply and accumulate operations every cycle. The interface with the processor uses ARM *AXI and AHB bus interfaces which might help bound the demand of the NPU on the shared memory, and the interference it generates.*

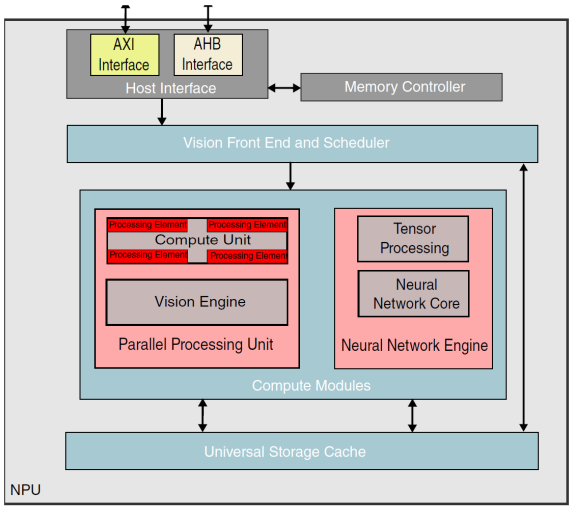

Fig. 10. NPU High-level Block Diagram in the i.MX 8M Plus processor [18]

It is difficult to model such a COTS device with no further information on its functional blocks, or without a characterisation by evaluation. It could be abstracted as a single initiator. This abstraction would need to be supported by limiting the use of the NPU as a unitary *accelerator, e.g. through platform configuration. Furthermore, the abstraction will still require an assessment of the nature and volume of transactions the NPU generates.*

The fourth case concerns *active* accelerators. An example of such accelerators are GPU.

Category 4. ***Active accelerator.*** *The accelerator operates independently and generates many* load *and* store *transactions.* ***Modelling impact on*** **PML.** *A unitary accelerator is thus modelled as a single initiator where, as for semi-active accelerators, parallel accelerators would need more refined analyses to check whether they will be decomposed into one or more initiators.*

Example 7 (of category 4)**.** *When the accelerator is a GPU used by a unique application at a time, it can be modelled as an initiator and single transaction forking to multiple targets should capture the combinations of behaviours of multiple threads running concurrently on the accelerator. Threads from the same application may not be considered as interfering with each other but with other applications in the system. The GPU scheduler decides upon execution of a computation kernel of the allocation of different blocks of threads to cores. The scheduling policy on most COTS platforms is subject to speculation, and the allocation of threads to cores is dynamic.*

In PML*, the initiator of a transaction from a given thread would thus be uncertain as well as for AMC20-193. Modelling the GPU as a single initiator abstracts away this uncertainty. This should be a conservative, but sound abstraction for interference analysis between applications. It needs to be backed by the platform to ensure only one task accesses the GPU at any given time.*

Example 8 (of category 4)**.** *When the GPU is used simultaneously by several applications, the GPU cannot probably be modelled as a single unit. Different threads from different applications may share the GPU cores, interfering on the GPU internal resources and the shared platform resources. Uncertainty may arise in the mapping of threads to cores, and thus the generated interference by an application.*

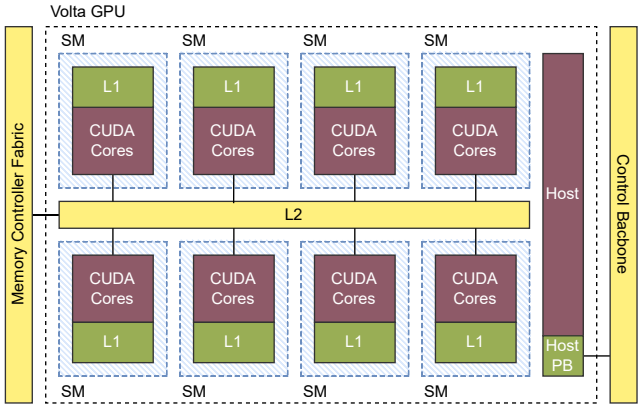

Fig. 11. Simplified PML model for the Volta GPU

However, the exact group of cores where an application is scheduled may not be relevant, provided said group is equivalent to the other groups of core on the platform. Capturing such platform symmetries in the PML *models would allow for some level of uncertainty. As illustrated in Figure 11, SM are symmetrical groups of cores on the Volta GPU (Example 2). Each SM has the same number of cores and private resources. Thus a group of threads should exhibit the same behaviour running in isolation in either SM. All SM can access the same shared resources through the same paths on the Volta; the interference suffered and generated by a group of threads is thus independent of the SM where they run. Isolating different applications to separate SM does however rely on undocumented support from the platform [19] (causing issues for Activity 3 in Section V-B).*

Example 9 (of category 4)**.** *A NVDLA in a "Large" configuration features its own separate microcontroller, depicted in Figure 12, tightly coupled with the accelerator. Where the CPU was in charge in the small configuration of Example 5, the microcontroller drives the accelerator. Modelling the whole as a single accelerator would fail to distinguish transactions originating from the microcontroller and ones originating from the NVDLA functional blocks. Each functional block of the*

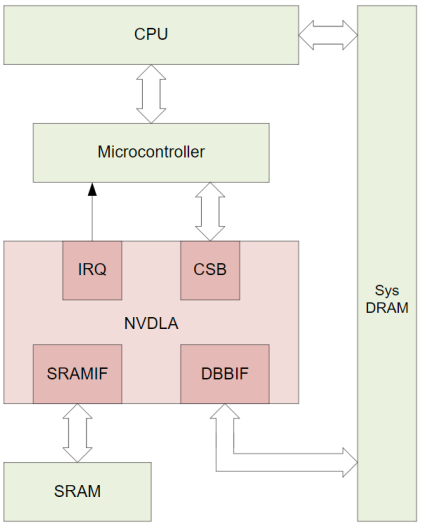

Fig. 12. Integration of the NVIDIA NVDLA in a large configuration [17]

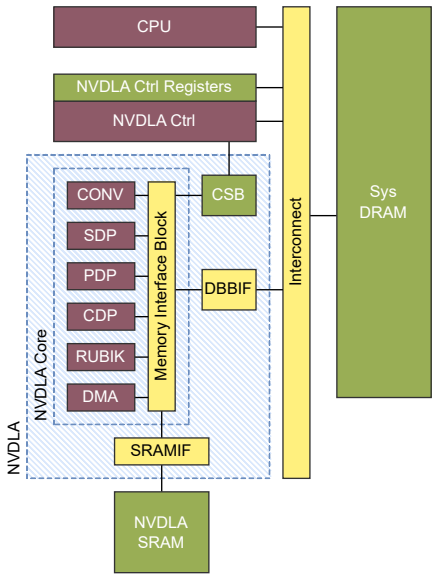

Fig. 13. Simplified PML model for the Large NVDLA

NVDLA can be mapped to its own initiator, as depicted in Figure 13. This abstraction, compared to the one in Example 5, would allow transactions where one or more applications use the different functional blocks without interfering. However each component (CONV, SDP, PDP...) may operate independently and interfere on the DBBIF.

Example 10. *The Xilinx ZYNQ-7000 AP [20], outlined in Figure 14, is a FPGA SoC with both Programmable Logic (PL) and Processing System (PS). The PS features a 2-core A9 processor, with a NEON VAU, memory resources, and inputs/outputs. The processor offers multiple ports to connect PL devices to resources on the PS. Different ports may reach different or the same resources, through different protocols. Depending on if and how PL devices use said ports, the ports themselves or devices on the PL side may become shared resources and be classified as interference channels.*

The PL features three types of ports: 4 general-purpose AXI ports (2 master and 2 slaves), 4 high-performance AXI master ports, and 1 AXI ACP port. The different ports first exhibit functional differences: as master ports cannot be used for the A9 processor to initiate reads from the PL. The AXI ACP port offers a high throughput and limited hardware coherency, as its accesses traverse the processor. However, it may result in serious cache trashing on the processor (as a result of invalidations), and interference on the A9 processor interconnect. The general-purpose ports allow access to most of the SoC interfaces, but share the interconnect with all input/output devices. The high performance ports only support high-throughput accesses from the PL to the main memory.

As a programmable logic device, the model for an FPGA is dependent on the devices and functions that have been configured, and on their use of the available platform resources. As an example, a DMA configured on the PL may solely read memory from the flash controller using a general purpose

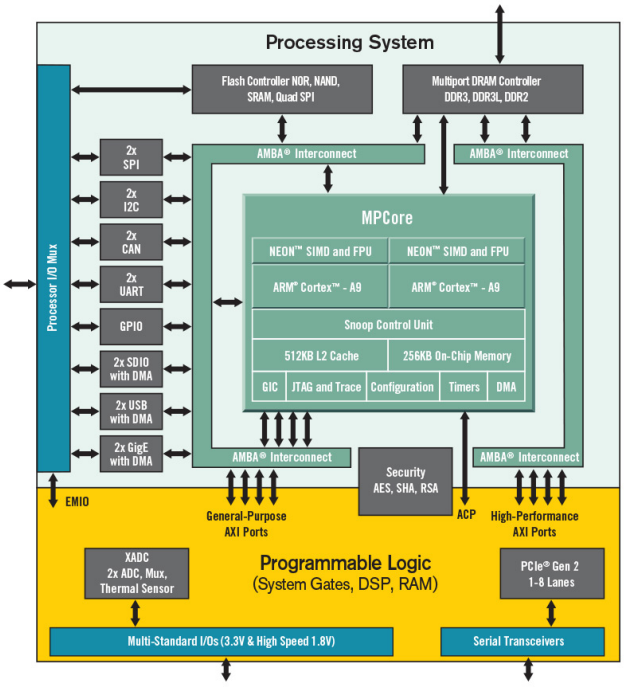

Fig. 14. Overview of the Xilinx ZYNQ-A7000 AP

port. It initiates transactions, contributing to and suffering from interference on shared resources. As such, it should be included as an initiator in the PML *platform model. Unless its interference is mitigated, it should further be included as part of the final system configuration during analyses and tests.*

Care is thus required upon integrating devices on the PL side. Each configured device should be considered and modelled per the aforementioned cases. The PS can be modelled

as any platform. Existing interfaces to the PL or between the PS and PL should also be considered as part of the model most likely as transporters, based on their use by configured devices.

D. Dimension 3

The third dimension concerns the applicative layers that necessarily come with the accelerator, e.g. a runtime used to offload work from the CPU to an accelerator. They contribute to the interference generated on a platform. As an example the scheduling queue for a device may be shared between different applications, causing delays depending on the scheduler. The transactions generated by an applicative layer also need to be characterised, by assessing their documentation and their use of resources on the platform. The identification and verification must include all software running on accelerators as well as software interfaces or libraries used to program them. Some accelerators may indeed only be addressed through vendor-specific software interfaces.

Example 11. *The definition and execution of kernels, functions running on the Volta GPU, use the CUDA toolkit, or use higher-level libraries and runtimes which themselves offload computation on the GPU through CUDA. CUDA Kernels are written using a superset of a subset of C/C++. That is kernel code supports most of the C language, and the toolkit provides additional syntax for mapping code and data to the GPU, or calling kernels. As such CUDA-enabled code cannot be analysed through existing tools as it may not parse as valid C/C++.*

As part of the CUDA toolkit are the compiler (`nvcc`) and assembler (`ptxas`) . The compiler is based on the mature LLVM compiler. The open-source nature of LLVM supports the verification of the generated code, and the development of compiler passes to support further analyses [21]. The assembler, which converts NVIDIA virtual assembly format into an executable binary, is closed. Information relevant for timing or coverage analysis may thus be lost at compilation.

Example 12. *NEON instructions can be exploited through compiler optimisations, intrinsics, or assembly code. Intrinsics are compiler- or vendor-provided functions often used to expose optimisations or vectorisation in languages without such constructs such as C. Compiler optimisations may jeopardize the traceability of the generated binary to the original source [21], and* ARM *recommends the use of intrinsics over manual assembly code. Intrinsics explicit the use of vectorisation and of the NEON VAU. The added benefit is that the source code only exposes function calls, amenable to analysis.*

Example 13. *The software stack for the NVDLA comprises at its core the User-mode driver (UMD) and the Kernel-mode driver (KMD). The UMD loads a representation of a neural network, maps its inputs and outputs in memory, and informs the KMD that an inference job is ready. The KMD schedules available jobs, allocating DNN layers to function blocks, configuring the NVDLA registers, and collecting completed jobs. The KMD (and UMD) can run on the main CPU ("Small" system in Figure 8) or through a dedicated core ("Large" system).*

Similarly the open source software stack clearly identifies all required software, and opens the source code for analyses such as coverage or timing. Note that the NVDLA itself does not feature a core which executes user- or vendor-defined software. A NVDLA-enabled platform, depending on the integration, may not fall under the multi-core processor classification. Nonetheless, it still counts as one or more initiators as, once configured through the CSB, each block may initiate transactions to the memory.

Example 14. *The NPU is accessed through an OpenVX Driver. OpenVX [22] is a standard and API which defines reusable computer vision and neural network functions. An OpenVX computation is expressed as a graph. Each node in the graph refers to its parameters and a kernel, the underlying function. The standard defines a number of vision and neural network functions. OpenVX is supported as a backend for numerous neural network runtimes through the Neural Network Runtime middleware [23].*

Nevertheless, the use of such runtimes raises several concerns. The transition from a model (computation graph) to software items is not explicit, and controlled by the runtime itself. This is not in line with the identification of software running on the platform as per AMC20-193. As the NPU supports only a subset of the OpenVX functions, runtimes may further elect to fallback to the CPU to run some software items. Using the NPU through the lower-level OpenVX driver would provide control over software items allocation between cores and the NPU. However, additional characterisation effort is still required to clarify the transactions the NPU might initiate.

VII. Related Work

Worst-Case Execution Time (WCET) analysis methods [24], [25], [26], [27] rely on accurate processor models to produce conservative timing estimates of the execution of applications on a processor. As such, the underlying processor models do often capture a more concrete and precise representation of the processor, e.g. accounting for the internal state of a core. Those are finer-grained models than our transaction-based approach, but validating the underlying models may be a complex process [28]. To the best of our knowledge, PasTiS [6] is one of the few efforts to build a GPU model.

PML takes inspiration from Initiator-Target modelling approaches found as an example in in [29], where paths to shared resources are paramount to the interference analysis. The computation of interfering paths exponentially grows as a function of the number of initiators and targets. To cope with this issue, they propose to introduce reduction criteria (e.g., symmetries).

(Memory) interference analysis approaches fall in two main categories: (1) Request-driven, which is based on a per-(memory) request analysis of an application [30], (2) job-driven, which focuses on the number of (memory) requests

of an application as a whole. Hybrid approaches blend the request-driven and job-driven [30], i.e. considering both approaches jointly in a analysis [31].

Model checking can be used to identify the interference of a platform as done in [32]. To do so, the approach uses formal languages for describing the behaviour of the application and multicore platform and introducing the interference concept and CADP toolbox to evaluate the model.

Interference mitigation techniques are used for minimizing, or even eliminating, the resource contention impact between processing cores. These techniques either make use of space (e.g., cache partitioning, bank parittioning) or time (e.g., scheduling, bandwidth reservation) partitioning to reduce the impact that interference entails. Survey [33] summarizes many of the techniques employed to this end.

VIII. Conclusion and perspectives

We discussed the impact hybrid platforms on certification objectives for avionic systems. Hybrid platforms embed several cores and accelerator devices in a small package, to provide high computational power while satisfying strict SWaP constraints. We considered in particular two AMC: AMC20-152A for airborne electronic hardware, and AMC20-193 for multi-core platforms. Both require careful consideration about how devices are used and integrated in the system.

Most accelerators support highly parallel workloads and as such fall into the AMC20-152A *complex* device category, and in scope of the AMC20-193. As such, they require a thorough assessment of their behaviour and their integration in the platform. We thus considered the use of PML to capture and model knowledge about said devcices. We identified 3 main dimensions relating to the hardware and software integration of the device in the platform, and proposed a related taxonomy.

We introduced a number of examples of COTS and Soft IP devices to illustrate the proposed taxonomy with PML modelling templates. COTS devices expose little information about their behaviour, and sometimes very limited control on said behaviour. They thus require conservative assumptions and abstractions to comply with certification requirements. Said abstractions have an impact on the performance of the accelerator and they do require backing by the platform configuration, e.g. a single GPU user.

On the other hand, Soft IP (or custom devices), such as the NVDLA, do provide extensive information about their behaviour. They also tend to offer higher configurability than COTS devices. However, they do require separate objectives per AMC20-152A. There might also be a vast amount of implementation and configuration choices to compare to select the most suitable integration w.r.t. to certification and performance objectives.

We did highlight that PML is generic enough to model complex accelerators. However, we also identified venues for improvements. Accelerators such as GPUs cause uncertainty about the allocation of applications (threads) to initiators (cores), and thus the source of transactions. The highly parallel nature of accelerators does also imply a high number of initiators in the system. This raises concerns about the required granularity of the platform model, the scalability of related analyses, and that of their output.

Acknowledgement

The work presented in this paper is part of the PHYLOG 2 project supported by the Directorate General of Civil Aviation (DGAC). It is funded by the French government through the France Relance program, based on the funding from the European Union through the NextGenerationEU program.

The work presented in this paper has been funded in part by the Agence Nationale de la Recherche (ANR) under project "ANR-22-CE92-0066-01".

References

[1] F. Boniol, Y. Bouchebaba, J. Brunel, K. Delmas, T. Loquen, A. Mascarenas Gonzalez, C. Pagetti, T. Polacsek, and N. Sensfelder, "PHYLOG certification methodology: a sane way to embed multi-core processors," in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.

[2] F. Boniol, J. Brunel, K. Delmas, C. Pagetti, and V. Jegu, "Modelling and analyzing multi-core COTS processors," in *11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, 2022.

[3] EASA, "AMC (Acceptable Means of Compliance) 20-152A Development Assurance for Airborne Electronic Hardware (AEH)," 2021.

[4] ——, "AMC (Acceptable Means of Compliance) 20-193 on the use of multi-core processors (MCPs)," 2020.

[5] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the NVIDIA volta GPU architecture via microbenchmarking," *CoRR*, vol. abs/1804.06826, 2018. [Online]. Available: http://arxiv.org/abs/1804.06826

[6] M. Adalbert, T. Carle, and C. Rochange, "PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications," in *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, Toulouse, France, Jun. 2022. [Online]. Available: https://ut3-toulouseinp.hal.science/hal-03684680

[7] N. M. Otterness, "Developing Real-Time GPU-Sharing Platforms for Artificial-Intelligence Applications," Ph.D. dissertation, 2022.

[8] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 213–225.

[9] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems," in *ECRTS*, 2018.

[10] T. Amert, "Enabling Real-Time Certification of Autonomous Driving Applications," Ph.D. dissertation, 2021, aAI28650154.

[11] A. Betts and A. Donaldson, "Estimating the wcet of gpu-accelerated applications using hybrid analysis," in *2013 25th Euromicro Conference on Real-Time Systems*, 2013, pp. 193–202.

[12] R. Pujol, J. Jorba, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. Cazorla, "Vector Extensions in COTS Processors to Increase Guaranteed Performance in Real-Time Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, jan 2023. [Online]. Available: https://doi.org/10.1145/3561054

[13] I. De Albuquerque Silva, T. Carle, A. Gauffriau, V. Jegu, and C. Pagetti, "A Predictable SIMD Library for GEMM Routines," in *2024 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.

[14] B. Lisper, "Towards Parallel Programming Models for Predictability," in *12th International Workshop on Worst-Case Execution Time Analysis*, vol. 23, 2012, pp. 48–58. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2012/3556

[15] NVIDIA, *NVIDIA Xavier Series System-on-Chip: Technical Reference Manual*, NVIDIA Corporation, Santa Clara, California, Apr. 2020.

[16] ARM, *ARM Cortex-A15 Technical Reference Manual*, 2011.

[17] "NVDLA Primer," http://nvdla.org/primer.html, accessed: 2023-06-28.

[18] NXP, *i.MX 8M Plus Applications Processor Reference Manual*, 2021.

[19] J. Bakita and J. H. Anderson, "Hardware Compute Partitioning on NVIDIA GPUs," *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 54–66, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259235797

[20] S. Ramagond, S. Yellampalli, and C. Kanagasabapathi, "A review and analysis of communication logic between pl and ps in zynq ap soc," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, 2017, pp. 946–951.

[21] H. Li, I. Puaut, and E. Rohou, "Tracing Flow Information for Tighter WCET Estimation: Application to Vectorization," in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015, pp. 217–226.

[22] Khronos Group, "The OpenVX Specification v1.3.1," https://registry.khronos.org/OpenVX/specs/1.3.1/html/OpenVX_Specification_1_3_1.html, accessed: 2023-06-28.

[23] NXP, *i.MX Machine Learning User's Guide*, 2020.

[24] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-case Execution-time Problem - Overview of Methods and Survey of Tools," *ACM Transactions Embedded Computing Systems*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[25] C. Ferdinand and R. Heckmann, "aiT: Worst-case execution time prediction by static program analysis," in *Building the Information Society: IFIP 18th World Computer Congress Topical Sessions 22–27 August 2004 Toulouse, France*. Springer, 2004, pp. 377–383.

[26] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "OTAWA: An Open Toolbox for Adaptive WCET Analysis," in *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP*, ser. Lecture Notes in Computer Science, S. L. Min, R. G. P. IV, P. P. Puschner, and T. Ungerer, Eds., vol. 6399. Springer, 2010, pp. 35–46.

[27] D. Hardy, B. Rouxel, and I. Puaut, "The Heptane Static Worst-Case Execution Time Estimation Tool," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*, vol. 57, 2017, pp. 8:1–8:12. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7303

[28] W.-T. Sun, E. Jenn, and H. Cassé, "Validating Static WCET Analysis: A Method and Its Application," in *19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019)*, Jul. 2019, pp. 6:1–6:10. [Online]. Available: https://hal.science/hal-02924072

[29] X. Jean, L. Mutuel, and V. Brindejonc, "Assurance methods for COTS multi-cores in avionics," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–7.

[30] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 145–154.

[31] M. Hassan and R. Pellizzoni, "Analysis of Memory-Contention in Heterogeneous COTS MPSoCs," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 165, 2020, pp. 23:1–23:24. [Online]. Available: https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2020.23

[32] V. A. Nguyen, E. Jenn, W. Serwe, F. Lang, and R. Mateescu, "Using Model Checking to Identify Timing Interferences on Multicore Processors," in *ERTS 2020 - 10th European Congress on Embedded Real Time Software and Systems*, Toulouse, France, Jan. 2020, pp. 1–10. [Online]. Available: https://inria.hal.science/hal-02462085

[33] T. Lugo, S. Lozano, J. Fernández, and J. Carretero, "A Survey of Techniques for Reducing Interference in Real-Time Applications on Multicore Platforms," *IEEE Access*, vol. 10, pp. 21 853–21 882, 2022.