SESP 2019
ARM Processors: Challenges for Next Generation Spacecraft Simulators

William Arrouy[(1)], **Christophe Duvernois**[(1)], **Denis Chatelais**[(1)]
Nadie Rousse[(2)]
Fernand Quartier[(3)], **Corentin Rossignon**[(3)]

[(1)]***AIRBUS Defence and Space***
31 rue des Cosmonautes
31402 Toulouse Cedex 4
Email: william.arrouy@airbus.com
christophe.duvernois@airbus.com
denis.chatelais@airbus.com

[(2)] ***Centre National d'Etudes Spatiales***
18, Avenue Edouard Belin
31401 Toulouse Cedex 4, France
Email: nadie.rousse@cnes.fr

[(3] ***SPACEBEL***
Ildefons Vandammestraat 7
Hoeilaart Office Park – Building B
1560 Hoeilaart, Belgium
Email: fernand.quartier@spacebel.com
corentin.rossignon@spacebel.com

ABSTRACT

The Space Industry has demonstrated that numerical processor emulation became an indispensable and a key element in spacecraft simulators. Besides an increasing use for on-board software development and validation, satellite system validation and operational use in control center, such simulators have now a major role in the overall spacecraft validation and operation.
Numerical emulators, in particular the current LEON based systems, have demonstrated their superior performance, flexibility, introspection and fault injection capabilities that cannot be matched by the real hardware, in the overall development cycle.
ARM based SoC (System-on-Chip) components and their ecosystems evolve at a blazing speed and dominate the low power embedded and mobile world. They start to carve their place in critical systems and in several space applications. Examples of COTS entering the space world are the SAMV71 (Cortex-M7 V7E-M)[1], Zynq SoC (Cortex-A9 V7-A)[2][3], the DAHLIA Cortex R52 [4] (ARM V8-R)[5], … They have not only the potential to multiply the performance and integration level of the current generation processors, they include a wealth of IO devices, substantial FPGA and adaptive reconfiguration capability along with multi-core and lockstep potential.
The wide processor and core choices and configurability, their sheer performance and complexity, the volume of attached IO and computing resources pose a tremendous challenge to the current mainstream processor emulation technology where fidelity remains critical.
Through a CNES study, AIRBUS and Spacebel are currently evaluating available ARM emulation solutions and products, the complexity of ARM processors and ARM simulation/emulation requirements with the objectives to find the optimal solutions, not only for a sole processor/On Board Computer but for a complex system such as a spacecraft simulator.
The first outcome of this study is showing the needs of new and innovative solutions to cope with the performance needs. It is also showing that the complexity does not only reside in the instruction set implementation or ARM processor mechanisms but on the numerous amounts of ARM board's specific devices/peripherals absent in the available simulation libraries and in the vendor products and tools.

The purpose of this paper is to present how the study group envisages this challenge providing an ARM emulation solution that can be integrated into wider simulation systems, which fulfills both the required performance and instruction timing fidelity while remaining modular enough to cope with the always evolving ARM chip structures and varieties. The paper will present the perspectives from classical emulation, emulation/execution on ARM architectures up to processor virtualization as a first step to para-virtualization.

Introduction

ARM architecture is of technical complexity level which may be considered much higher than the one of the SPARC processors currently used on space programs but also much problematic due to a variety of more than hundred Intellectual Properties (IP). Indeed, it is important to mention, as introduction that ARM only develops the processor(s) design and architecture which are then manufactured, under license, by a large number of other companies thus selling/providing a much larger amount of ARM based boards. The main consequence is that, depending on the board supplier/manufacturer varies the processor(s) micro-architecture(s) and the associated companion peripherals. In order to illustrate such complexity and variety can be non-exhaustively mentioned: multiple instruction sets (ARM, Jazelle, THUMB, THUMBEE …), SIMD instructions, predictors, advanced pipeline mechanisms, different cache policies and levels, several variations for memory management units or simply multi-core aspects; all implemented on various architectures starting from ARMv1 to ARMv8 or Cortex families A/M/R [6]. As a result, the ARM panorama is made of numerous processor models and obviously, variants. This large processor models amount and complexity is raising a completely new problematic for the development of Spacecraft (S/C) simulation.

ARM : A Vast World !

First and foremost, considering the large amount of ARM processors and architectures which kind of processors should we look at and target for our simulations? Regarding the literature and perspectives early 2016; mainly processors based on ARMv7 architecture (Cortex R for Critical Real-Time Application – Cortex A8/A9 for Applications) seem of interest. This is now mitigated by the start of DAHLIA project (Cortex R52) or more recent boards (UltraScale, UltraScale+ …)[7] based on the coupling of Cortex A53 and Cortex R5 processors. Apart from use of new processors, what becomes the major step two years later is the switch from ARMv7 to ARMv8 architecture. As a consequence, the study is focusing and considering the support: by order of complexity; of Cortex R4/5/7/8, Cortex A9, Cortex R52 and Cortex A53 with potential future extension to ARM Mali GPU. That is already a large challenge to support all of them.

Reuse or Redo ?

In a second step, do we have to develop our own emulation product or reuse existing product? To provide a pertinent answer a deep inventory and comparative study of available solutions/products on the market has been performed [8]. As outcome, a dedicated report under SpaceBel lead has been released. It appears that available solutions are quite numerous and plethoric (simple Google search quickly confirms); either from commercial or open-source products. Should be however noticeably mentioned, that despite cheap boards or supported debugging tools; virtualization and use of numerical simulation is still of interest to develop, execute and test software or for co-simulation purpose. Nevertheless, really stands out few commercial (ARM Fast Models [9], OVPSIm [10] …) and open-source offers; those ones been mainly QEMU [11] based. Commercial offers are quickly not considered for obvious reasons such as license costs, limited evolution capabilities, difficult integration in our simulator, used technologies, supported platform or more surprisingly non-availability of the targeted processors. More globally, most of the offer, despite numerous, does not match our needs and often suffers from a lack or reliability/fidelity (ARM Fast Models excluded).
Indeed, it should be kept in mind that our main objective is to develop S/C simulation. As a consequence the emulation shall fit the whole S/C simulator needs; optimizing its building, integration, performances, scalability and long term support/maintenance. On open-source side notably remains a very good candidate, namely QEMU. However, AIRBUS puts in balance; on one hand, the inheritance from past emulators development and on the other hand the effort to master QEMU internals, complexity of QEMU (especially to define an accurate timing model), QEMU technologies (especially outdated JIT engine) and potential evolutions foreseen to cope future needs or use cases. Finally, AIRBUS chooses to develop ARMADILLO [12]: its owned ARM emulation solution based and sharing common blocks with

SimLEON (AIRBUS LEON2/3 emulator). Similarly, Spacebel follows a similar path capitalizing on their large infrastructure, know-how and dynamic translation technology. Indeed, it is important to mention, again, that the emulator is developed and designed not to be a standalone emulator but as a product which can be easily integrated and which matches the need of a complex system such as a S/C simulator. It especially has to cover needs from on-board software validation up to satellite operations.
As a consequence, the specification of the capabilities an ARMv7 emulator shall have, has been released early 2017 in the frame of the study. This has been recently extended by ARMv8R emulator specifications to at least cope with DAHLIA SOC [13].

First Status

From the development aspects, it has been chosen to progress in a step by step approach from low to high ARM Architecture / Micro-Architecture complexity:

- ARMv7R – designed for deeply embedded / critical real-time applications
    - Instruction Set for Cortex R (limited to ARM and Thumb and limited compared to full v7 specification)
    - Cortex R4
    - Cortex R5/R7/R8, adding caches and predictors to target for example UltraScale boards and as preparation of R52
- ARMv7 – designed to save power / application with full O/S
    - Cortex A9 adding MMU and full ARMv7 ISA to target Xilinx Zynq 7000 boards [14] with execution of Linux as O/S.

Figure 1: Digilent 410-248 ZedBoard Zynq 7000 Development Board © Xilinx

- ARMv8R – Quite close from ARMv7 / critical real-time applications
    - Cortex R52 is adding ARMv8R new instructions and mechanisms and noticeably a new MPU. The management of multi-core aspects is also to be considered. It targets DAHLIA SOC which is one base for next generation On-Board Computers.

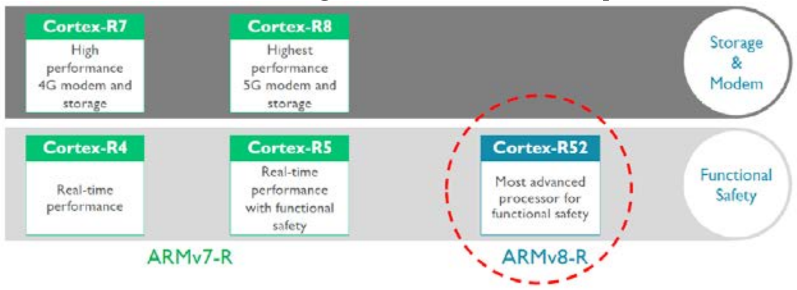

Cortex-R7
High performance 4G modem and storage
Cortex-R8
Highest performance 5G modem and storage
Storage & Modem
Cortex-R4
Real-time performance
Cortex-R5
Real-time performance with functional safety
Cortex-R52
Most advanced processor for functional safety
Functional Safety
ARMv7-R
ARMv8-R

Figure 2: Cortex R and Dahlia (© ARM)

- ARMv8 – Applications processors for feature rich OS and 3rd party applications
    - Cortex A53 [15] implements the full ARMv8 ISA, with 64 bits and evolution of the micro-architectures compared to v7 (MMU …). It targets Xilink UltraScale/UltraScale+ boards that may be complemented in the future by ARM Mali GPU emulation. Again Linux can be considered as a potential O/S.

It comes out that instructions handling (ARM and THUMB and thanks to RISC architecture), floating point simulation implementation and micro-architecture mechanisms are more complex but are remaining “classical” compared to LEON one(s). Furthermore, it should be pointed out that ARM ecosystem is accompanied with an important and complete documentation set. Further analysis, shows that major differences and difficulties are, as can be expected, located on the accurate timing model implementation due to cache policy, predictors and lack or at the contrary amount of non-intrusive debug traces. However, not fully anticipated, it also appears difficult, complex and time consuming, the simulation of the large list of devices placed around the processor(s) (GIC, SCI/LIN, RTI Timer, GPIO, DMA, Clock, CAN , SPI, Ethernet, SD Card, Flash Controllers, USB ….), whose most of them are required by the O/S or the applicative Software to execute properly. On top of that, devices definitions are neither fixed, nor standardized and change from one supplier to another or simply from one supplier board to another. It should also be pointed out, that this latter aspect has also been identified as a major drawback of open-source on (some) commercial solutions. Indeed, they are mainly focusing on ARM targets supporting Linux as O/S and preferably used by game consoles or mobile phones. As an important consequence, our development shall handle this aspect providing sufficient modularity, configurability and scalability allowing integration of several set of peripherals. Finally, should be mentioned that this is also coupled to potential endianness issues in the peripheral communication, this being configurable at processor level.

Performances: a key issue

From the prototyping activity and through execution of tests software (or coupled with others R&D and use of real-time O/S) comes also a second critical issue: the execution performances, in other words how fast the emulation can run compared to target hardware. Indeed, it starts becoming a real challenge simply looking at the envisaged ARM processor frequency or performant micro-architecture.

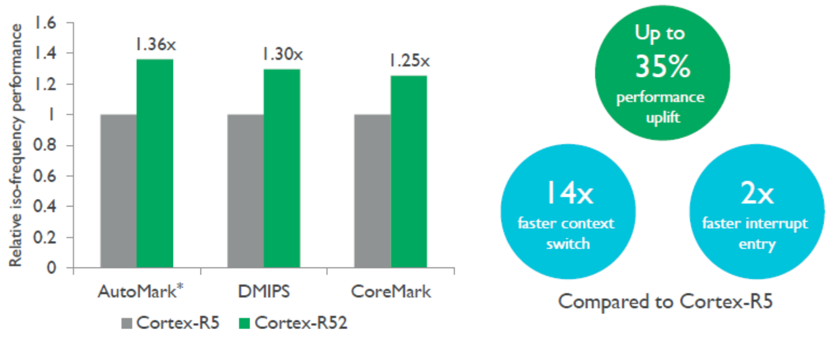


Figure 3: R52 compared to R5 Performances © ARM

Currently, ARMADILLO executes in real-time the emulation of an ARMv7 processor clocked to about 600 MHz. This figure is achieved taking full benefit of the static JIT engine and it is based on the agglomeration of a set of benchmarks. Moreover, performance comparison with other emulators hopefully shows similar or better performances than the one announced for instance by OVPSim (>1000 MIPS). Despite such performances; already estimated better than QEMU (foreseen to be compared as part of a CNES study); the current approach start reaching its limit and may impair the use of simulators as they are today. As depicted hereafter, it becomes a critical issue with DAHLIA SoC.

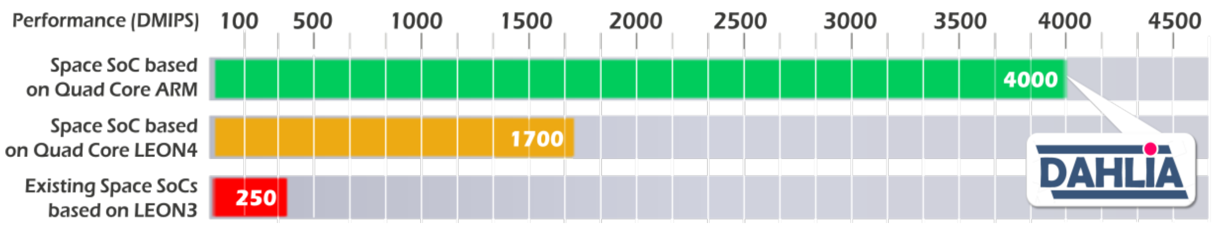


Figure 4: Dahlia Performances

SoC’s, such as Dahlia, include not only a large set of IP’s but equally European FPGA technology that can contain 500 kLUT’s (LookUp Tables) whereas LEON 4 needs 4 kLUT’s. The FPGA can contain space specific interfaces, such as

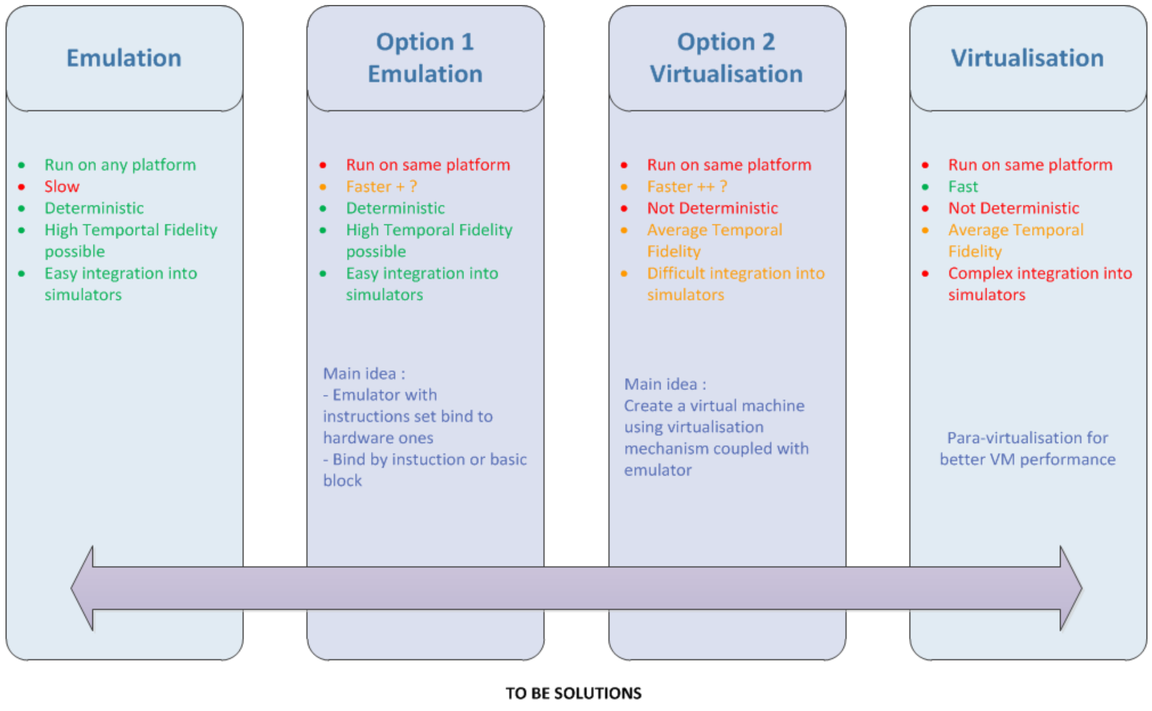
TM/TC and SpaceWire, specialised front-end, pre- and co-processors, and through their several serial lines of 6.25 Gbps or more, become potent communication nodes for a large variety of high speed sensors and communication networks. The communication load on memories and AXI interconnect/crossbar with QoS guarantees might become a major complexity while representative system simulators will be needed for the development of the FPGA logic. Unfortunately, such complexity tends to increase considering UltraScale+ boards with 1.5 GHz Cortex A53 processors or upcoming SoC's that have capabilities that surpass 15 to 30 times the processing capacity of the current LEON 3 processors. In complement, the very dense technology with much more functionality becomes problematic in terms of observability and fault injection, calling even more for complete and complex chip emulation. Finally, precise power and related temperature simulation, which might be more and more dominated by the IO activity, might become a challenge for nanosatellites

ARM Tools: What a big choice !

Additionally, ARM is widely use (several billions processors sold) and comes with a wide, mature and largely supported tooling (O/S, SDE …) thus maintained by a very large community. As a consequence such large tooling coupled with better CPU performance opens new perspectives, such as a convergence to commercial offers for instance for O/S, SDE or applying automotive/aeronautic standards such as for instance ARINC. This is the third challenge we are facing directly impacting our simulations. As a consequence, AIRBUS is currently setting up a Cortex A9 emulation (Zynq 7000) supporting the boot and execution of Linux Kernel as O/S; a good validation/benchmark for our emulation.

What's next: Native Execution to Virtualisation

As a consequence, it should be pointed out that new technologies and new ways of simulating ARM processors, with its numerous and complex peripherals/devices, performances and standard tools are now to be considered switching from classical emulation to native execution to native execution or much promising: virtualization.


Emulation
Run on any platform
Slow
Deterministic
High Temportal Fidelity possible
Easy integration into simulators
Option 1 Emulation
Run on same platform
Faster + ?
Deterministic
High Temporal Fidelity possible
Easy integration into simulators
Main idea : - Emulator with instructions set bind to hardware ones - Bind by instuction or basic block
Option 2 Virtualisation
Run on same platform
Faster ++ ?
Not Deterministic
Average Temporal Fidelity
Difficult integration into simulators
Main idea : Create a virtual machine using virtualisation mechanism coupled with emulator
Virtualisation
Run on same platform
Fast
Not Deterministic
Average Temporal Fidelity
Complex integration into simulators
Para-virtualisation for better VM performance
TO BE SOLUTIONS
Figure 5: Emulation to Virtualisation

As depicted in the above figure, ARM Emulation execution on ARM host instead of x86 one (Option 1) is envisaged as a potential performance accelerator. Indeed, it is expecting to mainly improve the binary translation performed by ARMADILLO JIT engine (similar ISA but less translation compared to x86). The determinism and fidelity of the emulation are preserved by such solution. On top of that, preliminary tests based on IU instructions have been performed, showing approach validity but performance comparisons still needs to be assessed. Moreover, it should also be noticed that it positively contributes improving the test coverage and pertinence of ARMADILLO instruction emulation. However, this solution is to be mitigated by two main elements:

- It is currently difficult to find ARM boards with sufficient memory to execute a full S/C simulation. However it tends to improve with planned ARM servers [16].
- The processing power of ARM processors compared to Intel/AMD ones is largely in favor of Intel/AMD. Indeed; the first ones are designed to save energy (dissipation of a few watts) whereas latest Intel/AMD cores are designed for performances (dissipation of more than 150W). Furthermore, “Instructions Per Cycles” and “Processor Frequency”; factors directly impacting the “Program Execution Time”, are also largely better on Intel/AMD.

Native execution of the Software instead of emulation is; for sure; a performance accelerator but this is also raising several already known issues. First of all, according to the simulator platform; it requires at least to re-compile (ARM to ARM) or to port (ARM to x86) the “Software”. Secondly, are obviously to be raised the issues of time management (how to control/stop/pause execution of this application inside the simulator) and memory handling (for example how to save breakpoint) coupled to the fact that the “Software” (i.e. running on the simulator) shall not directly access HW resources (registers …). Envisaged solutions can be summarized by those three ones:

- “Applicative Software” can be extracted and integrated to a Model. Execution tends to be deterministic and only remains at functional level. Indeed, memory layout and time control is not ensured. Moreover, it does not support access to HW or O/S resources. However, as a main advantage, this is not ARM specific and an alternate solution to “Intelligent Payload” modelling thus avoiding simulation of the “Software” applicative layer in charge of TC reception and TM emission.
- Complete “Software” can be executed in a cradle simulating the O/S or hypervisor calls. It requires to completely “stubbed” the O/S calls. However, it requires the O/S, first to be defined, its interfaces to be limited (in amount) and stables or better fulfilling a standard. Up to now, it is considered difficult to achieve due to O/S variability/on-going selection process. However, this should be envisaged in collaboration with “Software Development Team”.
- As a complement, especially if the “Software” to execute is a Linux [17] application dialoging with HW interfaces through Kernel Modules; native execution becomes a valid, pertinent and globally cheaper option. The time control can be; up to a certain extent, managed and the memory saving envisaged through O/S specific mechanisms. The main identified drawback is that “Software” may run as a standalone application (independent from the simulator) thus also preventing multi-instance execution.

Consequently, except if some conditions are met, native execution provides a local solution tightly coupled to the “Software” implementation. That is why; the main axis now considered by AIRBUS is to switch from Emulation to Virtualisation. More in details, this step requires a Type 2 Hypervisor in order to simply allow integration in our simulator. Hopefully, a large set of hypervisor already exists (commercial or Open-Source) where among them can be listed Xen, VirtualBox, ESX, KVM, Bareflank…

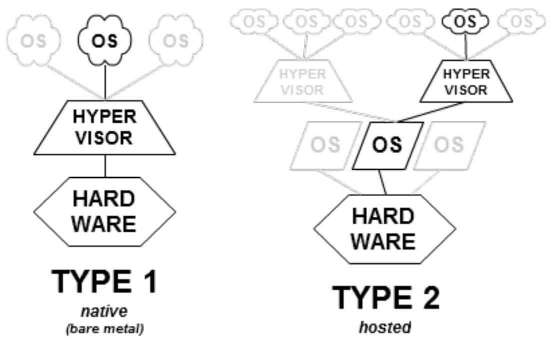


Figure 6: Hypervisors

In the frame of the CNES study, a prototyping is currently performed using KVM [18]. It has been mainly chosen due to its easy usage and relative simplicity (compared to others). However, from first usage, it quickly becomes clear that; in the future; a simpler hypervisor will better fit our needs; thus leading to the development of a custom hypervisor (the Virtual Machine) attached to ARMADILLO. On top of that, this custom hypervisor shall be able to execute on a Linux O/S without requiring tailoring of the Linux kernel (which at the end could be costly). The first outcome of the study, which is still on-going already, shows a list of technical issues to be tackled to allow integration into our simulators:

- Handling of external I/O – Memory (a solution has already been identified). It should be noticed that this especially requires support of Emulation for parts that cannot be handled by the VM.
- Handling of un-trapped privileged instructions (a solutions has already been identified)
- Execution of the Virtual Machine keeping Time Windowing principle (a solution has already been identified)
- Keeping Timing fidelity and determinism (some solutions have been identified and their limitation still need to be assessed).
- User/Kernel Space separation (prototyping is required mainly based on solutions implemented by VM such as VirtualBox)
- Simulation of the MPU/MMU (Shadow MPU/MMU) (prototyping is required for this latter topic).
- ARMADILLO needs to present a single product API which can be switched; without impact on the simulated models, either in interpreter, JIT and/or Hypervisor mode.

Finally, it is however a first step to virtualization with a long term objective to reach para-virtualisation in order to better optimize HW communication layer and continue gaining performances.

Conclusion

As a conclusion, this CNES study on ARM processor simulation highlights the fact that despite cheap H/W boards, numerical simulation offer is quite large and wide, thus showing a clear interest on its usage. However, except few commercial solutions and QEMU as open-source, products which can fit our needs are not so numerous taking into count requested fidelity, reliability, configurability, scalability and last but not least performances requirements. Furthermore, it remains important to focus on advanced/new technologies or way of thinking allowing us anticipating future increases in space processor performances at the time where comes to the panorama the DAHLIA On-Board computer based on a multi-core ARMv8R Cortex R52 processor. Moreover, this is to be complemented by the usage of Cortex A Family processors (A9 and A53) on Xilinx boards. On top of that, apart from performances, the next challenges with ARM simulation also encompass many features such as the variety of boards and associated peripherals but also a trend to use more commercial/standard protocols and tools thus, may be in the future; leading to a convergence with automotive methods and tools.
To tackle most of its problematic and in order to accelerate the execution speed of the processor simulation, a switch from emulation to virtualization is currently studied. However, it remains a first step to para-virtualization objective.

References

[1] ARM® Cortex M7 - https://developer.arm.com/products/processors/cortex-m/cortex-m7
[2] ARM® Cortex A9 - https://developer.arm.com/products/processors/cortex-a/cortex-a9
[3] ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C.c ARM DDI 0406C.c
[4] ARM® Cortex R52 - https://developer.arm.com/products/processors/cortex-r/cortex-r52
[5] ARM® Architecture Reference Manual ARMv8, for ARMv8-A profile edition Issue D.a ARM DDI 0487D.a
[6] https://en.wikipedia.org/wiki/ARM_architecture#32-bit_architecture
[7] Xilinx® Zynq Ultrascale - https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[8] Mike Noel, Corentin Rossignon, Fernand Quartier - Evaluation of ARM Processor Emulators - SPB-TN01-DOC-507 27-6-2018
[9] http://infocenter.arm.com/help/topic/com.arm.doc.dui0370n/DUI0370N_fast_model_ug.pdf
[10] Imperas - OVPSim http://www.ovpworld.org/technology_ovpsim
[11] QEMU - http://www.qemu-project.org/

[12] W.Arrouy, C.Duvernois, “ARM Processors Family Emulation” – SESP 2017 March, ESA-ESTEC, Noordwijk, The Netherlands
[13] Dahlia project - http://dahlia-h2020.eu/
[14] Xilinx® Zynq 7000 - https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf
[15] ARM® Cortex A53- https://developer.arm.com/products/processors/cortex-a/cortex-a53
[16] https://b2b.gigabyte.com/ARM-Server/Cavium-ThunderX2
[17] Xilinx® Linux - https://github.com/Xilinx/linux-xlnx
[18] Kernel-based Virtual Machine - https://www.linux-kvm.org/