ARM Processors Family Emulation

William ARROUY [(1)], Christophe DUVERNOIS [(1)]

[(1)]*AIRBUS Defence & Space*
31 Rue des cosmonautes
31402 Toulouse Cedex 4
william.arrouy@airbus.com
christophe.duvernois@airbus.com

ABSTRACT

Space industry requires performant and high fidelity processors emulation to simulate processors embedded in satellite on-board computer for on-board software development, validation or satellite operation. Up to the current LEON generation, simulator is cheaper and easier to use than hardware bench. Moreover, simulation provides, contrary to hardware, high level services such as failure injection, introspection or debugging which are largely used in the overall Avionic validation chain.

ARM processors and ARM based SOC components are nowadays widely used in numerous application domains such as mobile devices. As a consequence, exists on the market a multitude of SOC providers which includes ARM IP. ARM, recently starts to be used, or is envisaged, on space application programs.

AIRBUS is currently targeting ARMv7 [1] architecture emulation based on ARM Cortex cores. AIRBUS takes also part in studies with CNES and SPACEBEL related to ARM processor Cortex M series. Since ARM usage covers a very large and active community; a huge "ecosystem" and tools are available, obviously including several ARM instruction set simulators. However, after analysis, most of them only implement a subset of the CPU instructions set and/or are only targeting a specific SOC without modular design. On top of that, most advanced ones are usually not freely available and/or complex to use and with a shared disadvantage of limited/poor timing fidelity …

ARM architecture is of higher complexity compared to the SPARC one currently in used on space computers. Can be listed, for instance: multiple instruction sets (ARM, JAZELLE, THUMB, THUMBEE), Single Instruction Multiple Data instructions, pipeline and cache mechanisms or usage of multi-cores. Does this complexity impact our current JIT engine design? How multi-core technology can be integrated in the emulator design. What are the solutions to efficiently validate ARM emulators with respect to a wide range of already existing targets processors?

The purpose of this paper is to present how AIRBUS envisages this challenge providing ARM emulation solution that can be integrated into wider simulation systems, which fulfills both the required performances and instruction timing fidelity while remaining modular enough to cope with the always evolving ARM chips structures and varieties.

INTRODUCTION

ARM processors start been used or envisaged by Space Industry (AvionicX, ARM4Space, ASCOT, Ariane6 …) [2] as core processor for payload management, GNSS processing launchers, spacecraft monitoring and control processor... In the frame of such kind of programs AIRBUS uses to provide numerical simulation facilities for Operations but also for Software Validation campaign (main objective). This later activity especially requires advanced emulation features to first, achieve reasonable execution speed but also to commit on a high fidelity level with respect to target processor functional and timing aspects. To anticipate future coming space projects development or preliminary evaluation activities, it has been decided to quickly prototype ARM emulation solution for Software Validation (first focusing on ARMv7 but considering ARMv8 as possible extension); analyzing differences with well know and widely used in AIRBUS simulators LEON family, available open

source and commercial ARM emulators or hardware boards; all mainly focusing on functional/timing fidelity, speed and capabilities.

ARM VERSUS LEON PROCESSORS

The ARM architecture is Reduced Instruction Set Computer (RISC) architecture, basically incorporating for instance: a large uniform register file; a load/store architecture, where data-processing operations only operate on register contents and not directly on memory contents; simple addressing modes with all load/store addresses being determined from register contents and instruction fields only.

In order to achieve a good balance between high performance, small program size, low power consumption, and small silicon area; the ARM architecture also comes along with additional features such as:

- Instructions that combine a shift with an arithmetic or logical operation
- Auto-increment and auto-decrement addressing modes to optimize program loops
- Load and Store Multiple instructions to maximize data throughput
- Conditional execution of many instructions to maximize execution throughput.

Comparison between ARMv7; selected target processor family; and well known SPARC V8 (LEON) first shows that ARMv7architecture encompasses a wide processors series; one can split into three kinds of “profiles” each again derived in several versions [3]:

- A-profile: "Application" profile, especially implemented by Cortex-A 32-bit cores series. It is widely used in mobile devices. (smartphone, mobile computing, netbooks, servers …)
- M-profile: "Microcontroller" profile, mostly implemented by Cortex-M series. (engine, industrial control, flash drives …)
- R-profile: "Real-time" profile, mostly implemented by Cortex-R series (embedded processors for real-time signal processing, hard drive, mission-critical systems).

It has been chosen to mainly focus on R series with selection of Cortex R4 and R5 processors where cheap, very convenient and easy to use boards (for emulator characterization) are available compared to other Cortex profiles (e.g. Raspberry Pi and Cortex A7). More recent Cortex R series can also bet envisaged being supported, putting apart out of order pipeline execution. In addition, emulation support for A and M series is also implicitly considered by AIRBUS; as long as based on ARMv7 architecture; such as for instance Cortex-A5, A7 and A9 (targeting. Zynq [4]). On top of that, future emulation extension to support ARMv8 is also considered.

ARM processor has several CPU modes which can be switched due to interrupts, exception or programmatically, showing richer states compared to LEON. ARM uses a flat register model and has uniform 16× 32-bits registers including classical program counter, stack pointer and link register. Contrary to LEON, register windowing does not exist but register usage varies according to mode as depicted hereafter.

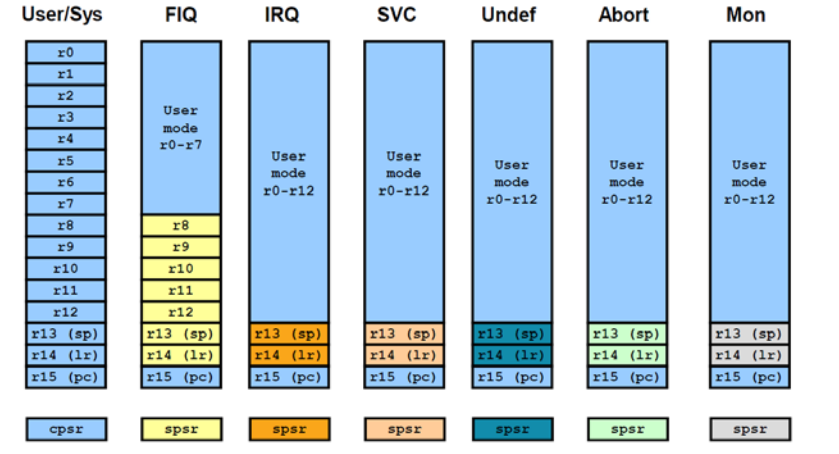

Fig.1. ARM registers and modes

The instructions set is composed ARM 32-bits completed with Thumb and Thumb-2. It should be noticed that emulation of Jazelle and Thumb-EE are put apart as no Java application and code generation at run time (JIT) are currently foreseen. Thumb is a 16-bits instructions set, designed to improve compiled code-density using shorter opcode, limited register access and reduced functionalities (e.g. only branches can be conditional). Thumb-2 extends the limited 16-bit Thumb with additional 32-bit instructions to give more breadth, thus producing a variable-length instruction set.

The ARM architecture provides a non-intrusive way of extending the instruction set using "coprocessors" which is addressed using MCR, MRC, MRRC, MCRR and similar instructions. The coprocessor space is logically divided into 16 coprocessors, coprocessor 15 (cp15) being reserved for typical control functions like caches and MMU/MPU operations.

Beside the ARM core, many ARMv7 SoCs are supporting separated data and instruction caches; pipelines with prediction algorithms (e.g. branch predictor and/or return stack predictor). Indeed, ARMv7 provides a cache coherent architecture designed for maximum flexibility: cache can scales from single CPU to massive Symmetric Multi-Processing (SMP) systems. On top of that, it is under SoC implementer configuration to select caches that are visible or not to software (or any points between those two options); to define multiple levels caches with separate I-cache and D-cache, specific write or cache line replacement policies (e.g. Round-robin or Pseudorandom); resulting in increased complexity for the definition of an emulator timing model.

Finally, ARM cores can alter the interpretation of word and half word data between big-endian and little-endian during runtime whereas LEON processor is big endian only.

DIFFERENCES LEON / ARM:

Although ARM shares RISC architecture with SPARC, it exists several differences between them. Indeed, ARMv7 implements powerful mechanisms leading to a generally more complex architecture than SPARCv8. Therefore it becomes a real challenge to develop an emulator as performant as SimLEON considering:

- Bit-accurate emulation: more complex instructions set thus including more capabilities.
- Cycle-accurate emulation: additional mechanisms and need to define a modular cache, pipeline, predictor models architecture in order to get efficient simulation and ability to quickly support new targets.
- Emulation speed: higher processor frequencies and more complex mechanisms impairing the challenge to meet at least real-time execution.

On top of that, since ARM does not produce chips but only IP, ARM cores are highly configurable. Consequently, it implies the availability of a wide range of CPU/SoCs on the market. As a result, in order to be able to efficiently support ARM processors emulation, it is mandatory to design a scalable and highly configurable solution on top of performance and accuracy needs.

NUMERICAL PROCESSOR EMULATOR: MAKE OR BUY?

ARM processors are widely used, for instance in mobile applications, resulting in wide ecosystem of compilers, tools and emulators (e.g ARMulator, SimIt-ARM, ARM-Sim, OVPSim, Simics, QEMU, RVISS, Keil …). An analysis of available ARM processor emulators shows that most of them only implement a subset of the ARMv7 processors series; often with poor scalability, emulate partially or with limitations the instructions sets, lack timing model or get licensing issues preventing updates and adaptations. On top of that, further envisaged evolutions to ARMv8 and multi-core support increase the problematic. As a consequence most of them are finally not fulfilling our needs with additional drawback of requiring important validation effort to get confidence on fidelity aspects. Only noticeable exceptions are QEMU [5] (open-source) and OVPSim [6] (commercial) which both require deeper and specific analysis. Previous and similar analysis has already been performed by AIRBUS regarding LEON family and QEMU with the conclusion that it does not stick to timing

and integration requirements resulting in the development of AIRBUS SimLEON emulator. Nevertheless, regarding ARM processors higher complexity compared to LEON, wide usage of ARM processors, availability of open-source solutions well community supported and maintained, widely used QEMU solutions or OVPSim impressive performances ; can be balanced in house development solution. However, still remains a major drawback with those products in the lack of cycle-accuracy (or closed to) model, as their goal is not timing fidelity but fast emulation thus implying shortcuts and limitations on the instructions processing. Moreover, additional features required for AIRBUS simulator needs, especially concerning Software Validation aspects are partially covered by QEMU or OVPSim. As a consequence, selection of QEMU or OPVSim would require significant efforts, first on acquiring detailed knowledge of those two emulator's internals, then on necessary adaptions and integration efforts (especially with OVPSim). It should however be mentioned that, those two solutions remain suitable and reliable when not fully functional/timing behavior is acceptable and fast/very fast execution is looked for.

WHY DEVELOPING OUR OWN EMULATOR?

Since an ARM development card is cheap (about $50 or less), easily available; why not using hardware board instead of numerical emulation?

Indeed, a huge range of ARM based board is available on the market with specific integration and usage implying additional efforts and cost when updated. On top of that, typical statement is that "Numerical Emulator can't beat Hardware when comes on the table bit and cycle accuracy needs". However our experience based on SimLEON (AIRBUS LEON2/3 Emulator) shows that reaching high timing accuracy is feasible thus keeping a high level of configurability (memory timings, replacing SDRAM by Flash …) which is difficult to achieve on hardware board based solutions. Other major point to consider is hardware obsolescence; especially with the ARM technologies mainly driven by the fast evolving telecom mobile market. This has already been faced using several cheap ARM boards (to perform preliminary tests on M3 and M4) that quickly became discontinued.

On top of that, the purpose of simulator developed by AIRBUS is to perform On Board Software Validation or Operations Preparation based on integration of numerical models representing Equipments and Space Environment. This point raises one of the major drawbacks of the hardware board solution when hardware has to interface numerical models. Interfacing hardware/numerical worlds may quickly become tricky on such processor architecture since for example requirements to meet real-time execution cannot be by-passed. On top of that, hardware interface to model could be intrusive (e.g. stop processor execution to wait for model response) potentially causing loss of accuracy. However, this complexity has already been assessed by AIRBUS, with a successful outcome, on the LEON SVF [7] study where a LEON3 board has been connected to numerical models.

Despite huge tool set, hardware processor is far less flexible than numerical one concerning services such as debugging, non-intrusive memory inspection, non-intrusive code coverage, failure injection, detection of non-expected operations such has falling into processor / co-processor bugs or floating point denormal numbers detection... Moreover, tests reproducibility is better ensured by numerical solutions compared to hardware one; especially on architecture where random cache policy and predictors are used.

AIRBUS has successfully developed, for many years, a large suite of processor emulators based on different architectures. They are tailored and configurable to target use-cases where fidelity and/or execution speed are mandatory. As a consequence of those past developments and in complement to acquired knowledge, AIRBUS has developed a framework (CESIUM) in order to optimize emulator development and configurability, which also natively supports binary translation capabilities thanks to up-to date LLVM JIT [8] engines with the purpose of improving execution performances.

As a result, thanks to AIRBUS experience and available framework/tools which has been put in balance with off-the -shelve ARM emulators availability; but efforts to acquire expertise on their design, integration in wider systems, adaptations required to achieve required functional fidelity, timing model injection (which naively

looks simple compared to instructions simulation but which at the end turns out to be of higher complexity), user services implementation and validation; it has been chosen instead to develop; based on AIRBUS large heritage libraries CESIUM; Armadillo ARM emulator.

ARMADILLO

ARMADILLO development recently starts first targeting TMS570LS2021/TMS570LS3137 [9] [10] [11] [12][13] (ARM Cortex R4) and then TMS570LC4357 [14] [15] board (ARM Cortex R5) for processor emulation running on both Linux 64-bits and standard Windows PCs. The development is first driven by the needs of functional and timing fidelity (for Software Validation) and then execution speed (potentially for Operation Simulation); keeping in mind the modularity and scalability objectives based on our already existing CESIUM framwork.

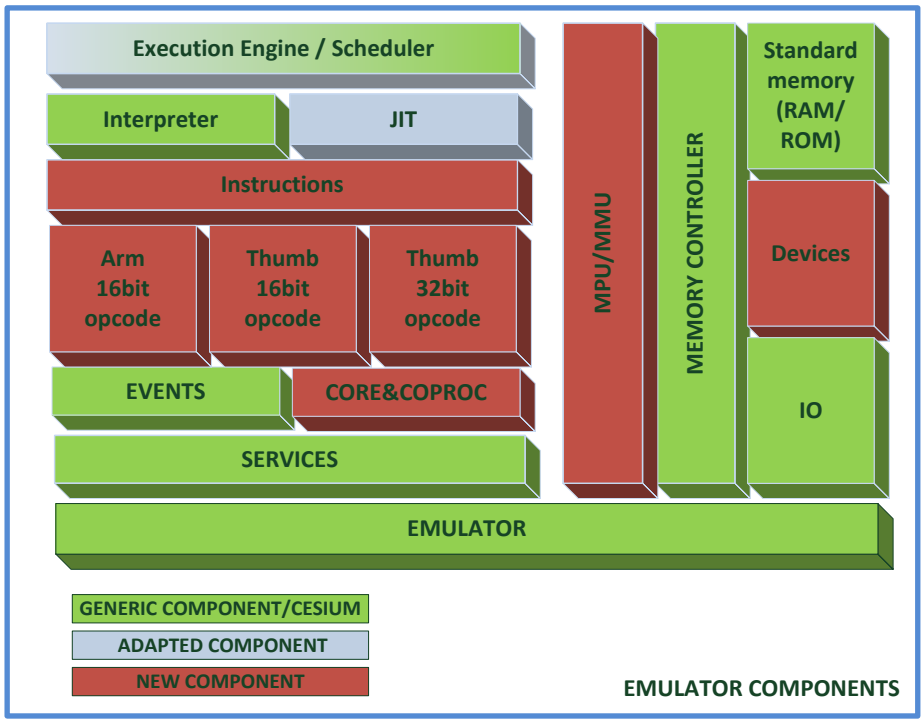

Fig.2. CESIUM and ARMADILLO components

Despite ARM architecture is RISC, Cortex processor is using variable length instructions (ARM/THUMB). As a result, the generic execution engine used for LEON emulators has first been updated and optimized. In a second step, the generic" Just In Time" engine has been strongly adapted to support more up-to date LLVM JIT core engine and to cope with the fact that instructions set can be selected at run-time, thus making complex/preventing static processing and optimizations before loading a software. Significant efforts has then been put on supporting efficient on the fly dynamic binary translation while software is emulated in order to achieve at least real-time execution considering 300MHz processor emulation.

Afterwards, obvious activity concerns the development of ARM and THUMB/THUMB-2 instructions sets for the decoding and execution of each opcodes (about 400) thanks to a generic interpreter block. This phase is immediately followed by a deep validation campaign based on comparisons against three Texas Instrument boards: TMS570LS2021, TMS570LS3137, and TMS570LC4357; where are especially considered, instructions functional behaviors but also effects of instructions combinations and sequencing. It should be mentioned that nearly all ARMv7 instructions contain a condition field which determines whether the CPU will execute them; having on one hand, the advantage for the programmer, to allow very dense in-line code without branches; but on the other hand, for the emulation, to perform significant additional operations directly impacting performances. On this point lies an important future improvement managed by others emulators relying on

similar Intel instructions functionality whereas ARMADILLO performs its own checks. However, this optimization requires a deeper analysis in order to keep functional fidelity level as discrepancies between Intel and ARM behavior exist.

In addition, ARM instruction set supports connection of up to 16 co-processors which is derived on emulator scalability and configurability requirements in order to easily plug any co-processor type. ARMADILLO achieves this integration at compiling time rather run-time or by abstraction of communications layers. Indeed, because the co-processors instructions are interleaved with ARM/THUMB ones, this aspect becomes key driver for performances. Among co-processors should be also pointed out FPU type based on Vector Floating Point technologies and Control Coprocessor CP15 to drive overall system control and configuration, memory protection unit, cache, etc...

Finally communication and interfaces devices, specific to ARM boards have been developed, covering a minimal set covering Timers, UARTS, and Interrupt Controllers, in order to be able to use emulator as a standalone product providing minimal multitasking and output capabilities. Memory management strongly relies on CESIUM existing components complemented by Memory Management and Protection Units, two elements strongly impacting the performances. Devices and memory layout are elements which significantly vary across boards. Similar solution as co-processor one has been chosen to configure the emulator at compiling time thanks to JSON configuration files. Those configuration files are not only used to define the target processor characteristics and memory mapping but also to control emulator automatic code generation where can be selected services, devices and/or functions to include. As an example, it allows generating one emulator binary tailored for timing and functional fidelity with a specific memory mapping, or configured to optimize execution speed (e.g. statistical timings, no breakpoints …). As a consequence, the target binary file becomes board and usage specific whereas the code remains generic, readable and tailorable.

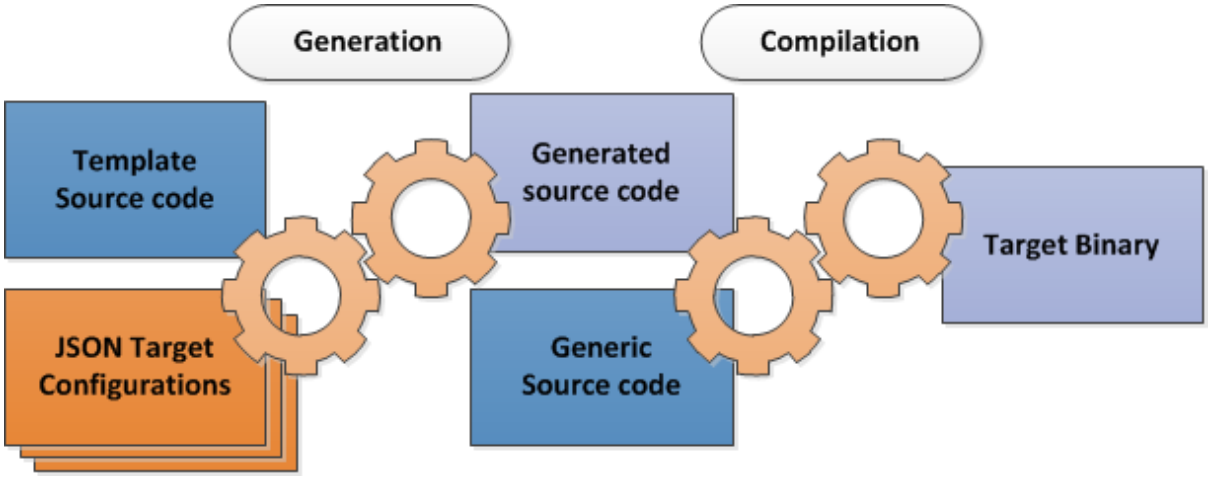

Fig.3. Emulator build process

ARMADILLO STATUS

ARMADILLO currently supports TMS570LS2021, TMS570LS3137, and TMS570LC4357 boards already emulating complex software such as complex boot software, hypervisor, RTEMS as well as a wide set of processors benchmarks increasing its confidence on maturity and fidelity level. Thanks to preliminary optimizations, real-time execution performance objective (for Software Validation goal) is already met on a 300MHz CortexR4 and in some extend on a Cortex R5. Performance increase is expected in the future thanks to better management and improvement of the binary translation mechanism and MPU; however, “will it be sufficient for Spacecraft Operations?” is still an open question.

Next remaining challenge is to tune ARMADILLO timing model; which takes into account instruction timing effects of the IU/FPU Pipelines, Instructions and Data L1/L2 caches, Branch and Stack Return Predictors; all those mechanisms been well known thanks to past experiences (SimLEON timing model [16]) or abundant documentation.

On top of that, thanks to the inherited CESIUM libraries a wide range of services (breakpoint, watchpoints, failure injection, events and scheduling management) are available including root elements to support multi-core configurations. Indeed, thanks to previous activities focusing on GR712 (LEON3-FT multi-core) [17] board emulation and optimizations; have been developed mechanisms and solutions to efficiently implement multi-core processor emulation. ARMADILLO architecture and CESIUM core libraries already natively embed such capability which will ease next step to ARMv8 processors such as the multi-core Cortex R52.

REFERENCES

[1] ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C.c ARM DDI 0406C.c

[2] O.Notebart – Reasearch and Technology activities in on-board data processing domain – ADCSS 2015 Avionics Technology Trends Workshop ESA/ESTEC in Noordwijk, The Netherlands, October 21st 2015

[3] https://en.wikipedia.org/wiki/ARM_architecture#32-bit_architecture

[4] https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[5] http://www.qemu-project.org/

[6] http://www.ovpworld.org/technology_ovpsim

[7] G.Queré, Poul Houggard, M.Caleno, OBC Simulator Architectures and Interfaces to System Test Benches / LeonSVF, TEC-ED & TEC-SW Final Presentation Days, December 2014

[8] LLVM (http://llvm.org/)

[9] TMS570LS20x/TMS570LS10x Technical Reference Manual SPNU489C - February 2012

[10] TMS570LS21x/TMS570LS31x User's Guide SPNU511D – November2014 - Revised December 2015

[11] TMS570LS21x/TMS570LS31x Technical Reference Manual SPNU499B - November 2012 – Revised August 2013

[12] Texas Instument TMS570LS2021 - SPNS141F– AUGUST 2010 – REVISED JULY 2011

[13] Texas Instument TMS570LS3137 - SPNS162C – APRIL 2012 – REVISED APRIL 2015

[14] Texas Instument TMS570LC4357 - SPNS195C – FEBRUARY 2014 – REVISED JUNE 2016

[15] Texas Instument TMS570LC4357 - SPNU563 – May2014

[16] W.Arrouy – C.Duvernois, "Methodology for timing characterisation of a LEON3 Numerical Emulator" SESP 2015 March, ESA-ESTEC, Noordwijk, The Netherlands

[17] http://www.gaisler.com/doc/gr712rc-board.pdf