Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D9.6– Space Applications Final Report**	
Work package	WP9	Space Applications
Task / Use Case	T9.1-T9.5	Space Applications
Subtasks involved	UC3.1-UC3.5	
Lead contractor	Infineon Technologies AG Dr. Werner Weber, mailto:werner.weber@infineon.com	
Deliverable responsible	Thales Alenia Space Spain Tres Cantos, Madrid	
Version number	v1.0	
Date	07/04/2017	
Status	Final	
Dissemination level	Public (PU)	

Authors

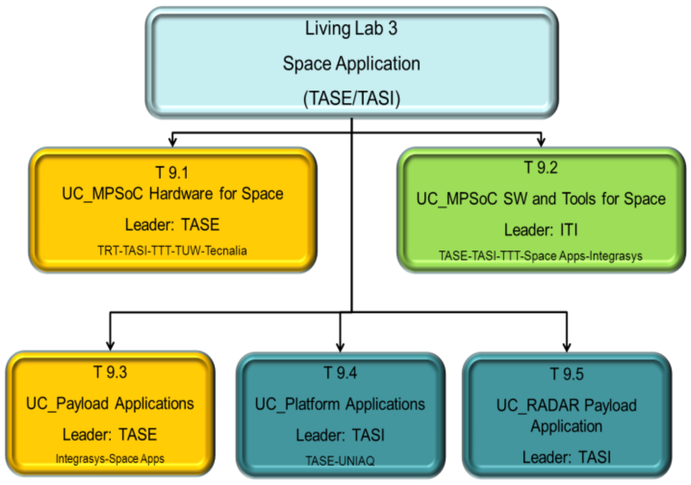
Partici-pant no.	Part. short name	Author name	Chapter(s)
64	15I INTEGRASYS	Juan Luis Mañas	Use Case Optical Payload Applications
20	02E TUW	Haris Isakovic	Use case MPSoC Hardware for Space
62	15E TECNALIA	Daniel Mugica	Use case MPSoC Hardware for Space
70	15Q ITI	Sergio Saez	Use Case MPSoC Software and Tools for Space
19	02D TTT	Arjan Geven	Use case MPSoC Hardware for Space
63	15F TASE	Dr. Manuel Sanchez	Use Case Optical Payload Applications
50	11J TASI	Dario Pascucci	Use Case Platform Application/ Radar Payload

Document History

Version	Date	Author name	Reason
v0.1	27/03/2017	WP9 Partners	First version
…			
v1.0	06/04/2017	Manuel Sanchez	Project internal review comments integrated, deliverable finalized
	07/04/2017	Alfred Hoess	Final editing and formatting; deliverable submission

Publishable Executive Summary

Scope of this document is to provide a detailed description of Space Applications in the frame of WP9 Living Lab. The figure below provides an overview of the WP9 organization. Target of this document is to support T9.1- T9.4 by the structuring of the use case building blocks.



WP9 Organization

This deliverable provides an explanation of the use cases and the contributions of the EMC² WP9 partners.

Table of contents

List of figures

List of tables

1. Introduction

1.1 Objective and scope of the document

Scope of this document is to summarize the space use case results of the demonstrators related with WP9 (Space Applications).

The description comprises the hardware and software involve in each use case. MPSoCs suitable for space applications have been selected in order to obtain fault-tolerance processors.

1.2 Structure of the deliverable report

The document is organized as follow: Section 2 provides the use case of hardware for space. Section 3 provides the use case of software and tools for space, while Section 4 provides the use case of optical payload applications. Finally Section 5 provide use case platform applications.

2. MPSoC Hardware for Space

2.1 Fault-Tolerant and Self-Healing Dynamic Reconfiguration Manager

A Reliable and Self-Healing Dynamic Reconfiguration Manager (DRM) is the solution proposed by TASE and TECNALIA in order to perform partial reconfiguration of a Virtex-5QV FPGA in a safe (reliable) way in the space environment, which is one of the main technical breakthroughs pursued in this WP9 use case. This solution has been designed in the context of WP4, while its robustness has been evaluated in WP9. A basic DRM, that is, a DRM without fault-tolerance elements, running on the Virtex-5 FPGA of the LADAP board and performing partial reconfiguration under command of LEON processor was demonstrated during the second year review. Now, all the specified fault-tolerance elements have been included to make the DRM reliable and self-healing and the final implementation has been tested and evaluated.

Figure 1 shows the final architecture block diagram of the Reliable and Self-Healing DRM designed by TECNALIA and implemented in the Virtex-5 device of the LADAP platform. It is externally controlled and monitored by a LEON processor implemented by TASE in the RTAX device of the hardware platform.

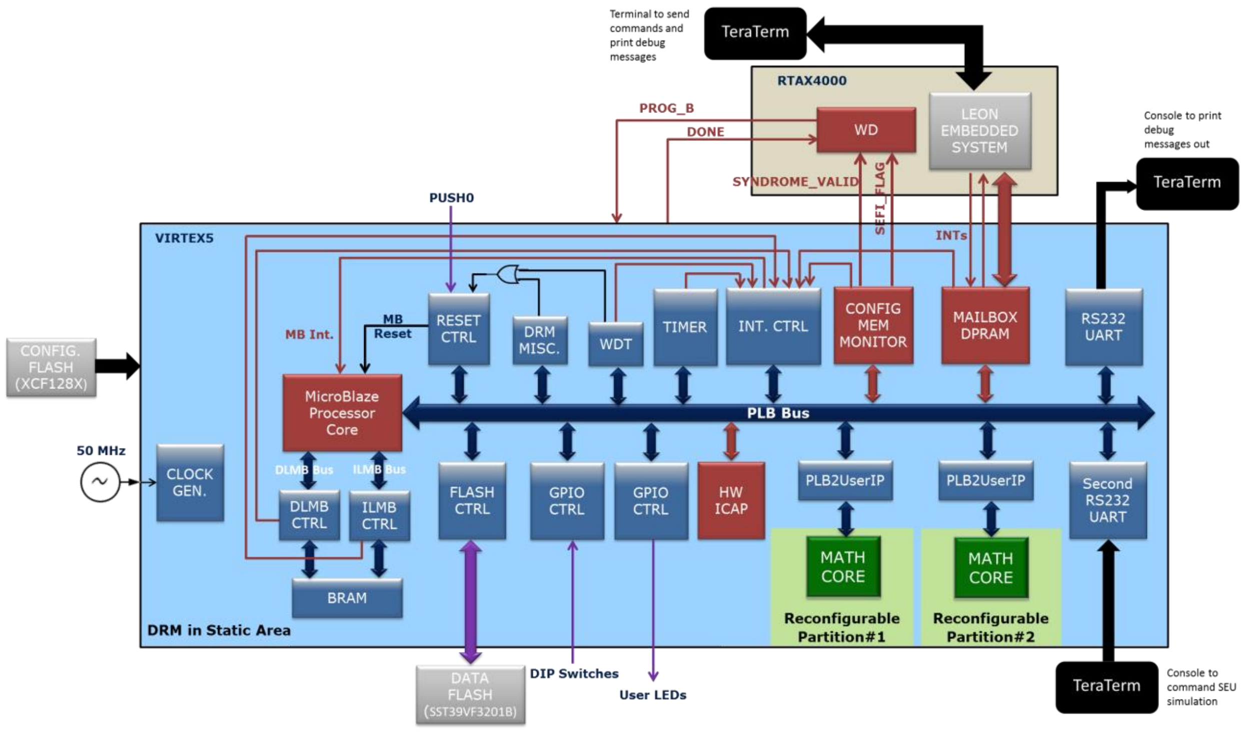


Figure 1: System architecture diagram of the Reliable and Self-Healing DRM

The DRM hardware implementation details have been described in D4.9 and D4.10, while the DRM software implementation details were explained in D9.5. The DRM functional validation tests and their results have been explained in deliverables D4.9 and D4.10. The rest of this section describes the DRM performance achievements and presents the evaluation of the final solution from the point of view of its robustness and self-healing capability.

2.1.1 DRM design performance

2.1.1.1 Scrubbing actions timing

The DRM performs three types of scrubbing actions:

- Scrubbing a configuration frame with one bit error. MicroBlaze reads the frame through the ICAP port, toggles the error bit and writes the corrected frame back.
- Scrubbing a configuration frame with two bit errors. Since the error positions are unknown, MicroBlaze gets the good configuration frame data from the external data flash and writes it into the FPGA configuration memory through the ICAP port.
- Full device scrubbing. MicroBlaze re-writes all the configuration frames of block type 000. As in the previous case, good configuration data is read from the data flash. This scrubbing action is performed when the Configuration Memory Monitor peripheral detects a global error in the configuration memory, or when periodical blind scrubbing is enabled (under LEON control).

The time that the DRM needs to perform those actions is shown in the next table. Details on the time needed to execute some of their steps are also provided. Note: these times have been measured when the DRM runs an application that does not print any message out to the Tera Term console, as it would be the case in a real scenario.

Parameter	Time
Time that MicroBlaze needs to read one configuration frame through the ICAP port	30.3 µs
Time that MicroBlaze needs to write one configuration frame through the ICAP port	33.7 µs
Time that MicroBlaze needs to scrub a frame with one bit error (configuration frame is read, error bit is toggled and frame is written back)	66.5 µs
Time to perform FAR register checking	16.4 µs
Time to perform fake SEFI checking	5.55 µs
Time from the MicroBlaze application main loop detects that there is a pending scrub action request (scrub_cmd_req $\neq$ 0) to the action is completed and the scrub_cmd_req flag is cleared, for the case of a frame with one bit error	**90 µs**
Time that MicroBlaze needs to scrub a frame with two bit errors (access to external data flash is required)	82.2 µs
Time from the MicroBlaze application main loop detects that there is a pending scrub action request (scrub_cmd_req $\neq$ 0) to the action is completed and the scrub_cmd_req flag is cleared, for the case of a frame with two bit errors	**105.6 µs**
Time to execute the *microblaze_scrub()* function	820 ns
Time that MicroBlaze needs to perform full device scrubbing	**1.176 s**
Time from Configuration Memory Monitor interrupt to the MicroBlaze application main loop detecting that there is a pending scrub action request (scrub_cmd_req $\neq$ 0)	**Variable from 2.8 to 3.6 µs aprox.**

Table 1: Scrubbing time measurements

When a SEU flips a configuration memory cell of the FPGA, the time that the Configuration Memory Monitor peripheral needs to detect it depends on the position of the flipped bit. This time can be anything from almost immediately to a maximum of one scan of the device. For the XC5FX130T device and a 50 MHz ICAP clock, the maximum time is equal to **22.01632 ms**.

Once the error bit is detected, and according to the time values shown in Table 1, the DRM will be able to correct it in less than **95 µs**. As reference, the SEU Controller macro provided by Xilinx will normally be able to correct the error in less than 250 µs. So the DRM is more than twice as fast as the Xilinx SEU Controller, which is a very good result.

2.1.1.2 DRM monitoring timing parameters

The watchdog timer implemented in the RTAX device has to monitor the SYNDROME_VALID output generated by the DRM Configuration Memory Monitor peripheral. If this signal does not have high pulses during some time, it is an indication of design intrusive SEFI. Under normal conditions, that is, in the absence of configuration memory errors, the SYNDORME_VALID signal has a high pulse every 41 or 49 ICAP clock cycles. When some error(s) is/are detected, the DRM performs the corresponding scrubbing action and the SYNDROME_VALID pulses stop for some time. The **maximum time without SYNDROME_VALID pulses is 1.176 seconds** (see Figure 4) which is the time needed by the DRM to perform full device scrubbing. Note: when the DRM runs an application that prints debug messages out to the Tera Term console, this time is slightly longer (1.182 seconds).

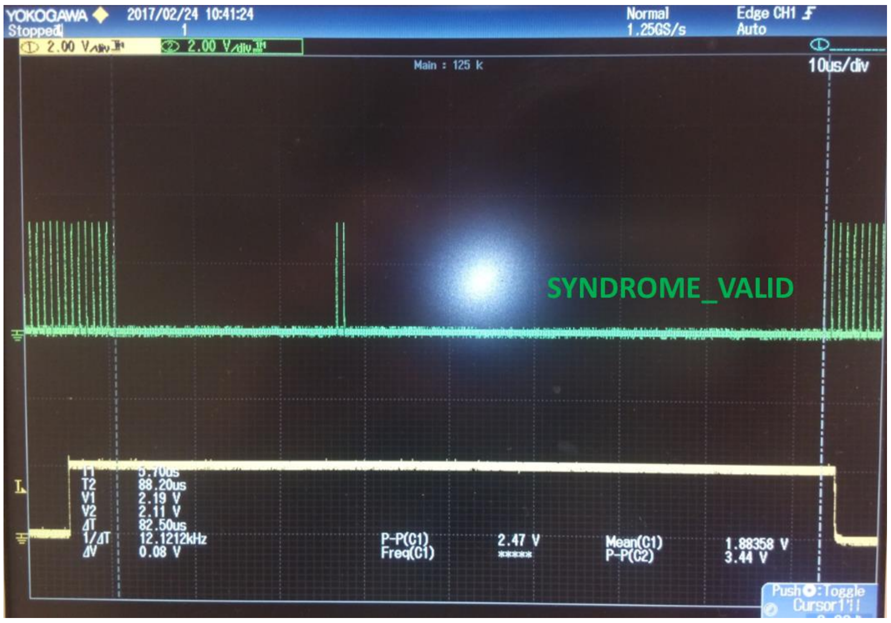


Figure 2: SYNDROME_VALID signal while scrubbing a configuration frame with one bit error

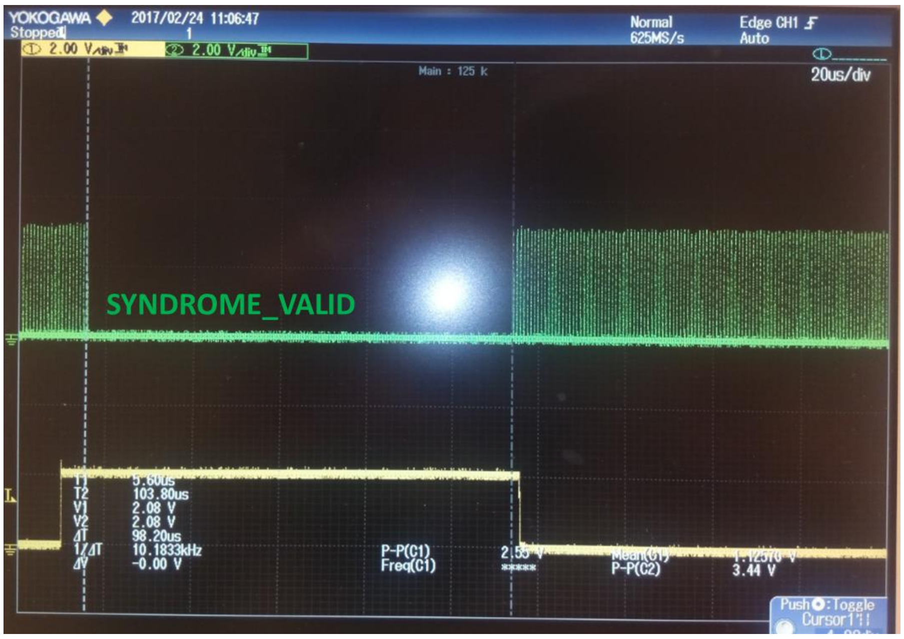


Figure 3: SYNDROME_VALID signal while scrubbing a configuration frame with two bit errors

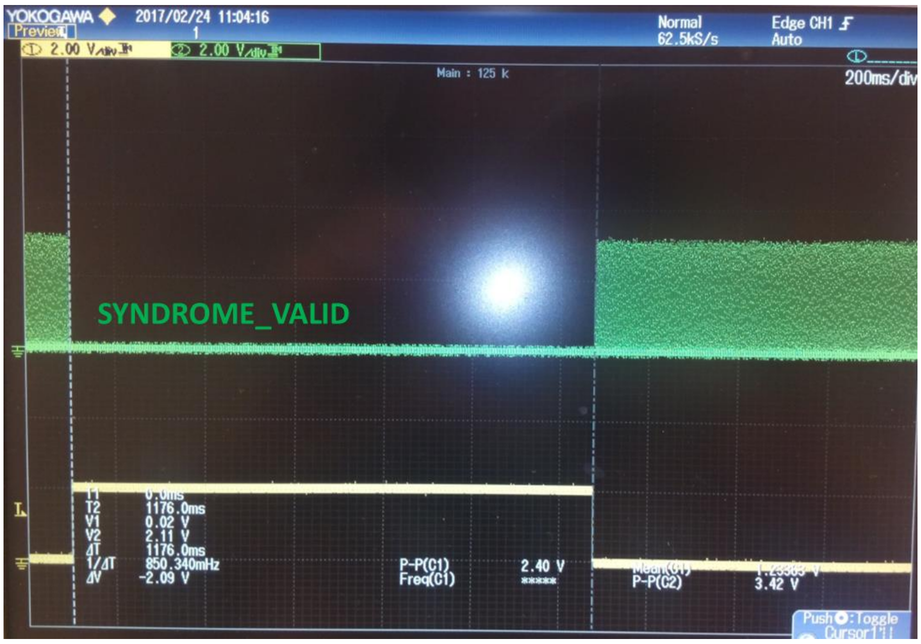


Figure 4: SYNDROME_VALID signal while full device scrubbing

The watchdog timer implemented in the RTAX device will pulse the PROG_B signal low for 280 ns, and the Virtex-5 FPGA will be reconfigured, when it detects that the SYNDROME_VALID pulses stop for more than 1.176 s (design intrusive SEFI symptom), when the DRM asserts the SEFI_FLAG (visibility loss SEFI symptom) or when the LEON processor requests the watchdog timer to reprogram the Virtex-5 device. During Virtex-5 reconfiguration, the DONE and SYNDROME_VALID signals are low. Reconfiguration ends successfully if the DONE signal goes high (this should happen 65.1 ms after the PROG_B falling edge, for the Virtex-5 master selectMAP configuration mode) and the SYNDROME_VALID pulses start again. The first SYNDROME_VALID pulse appears 1.06 µs after the DONE signal goes high.

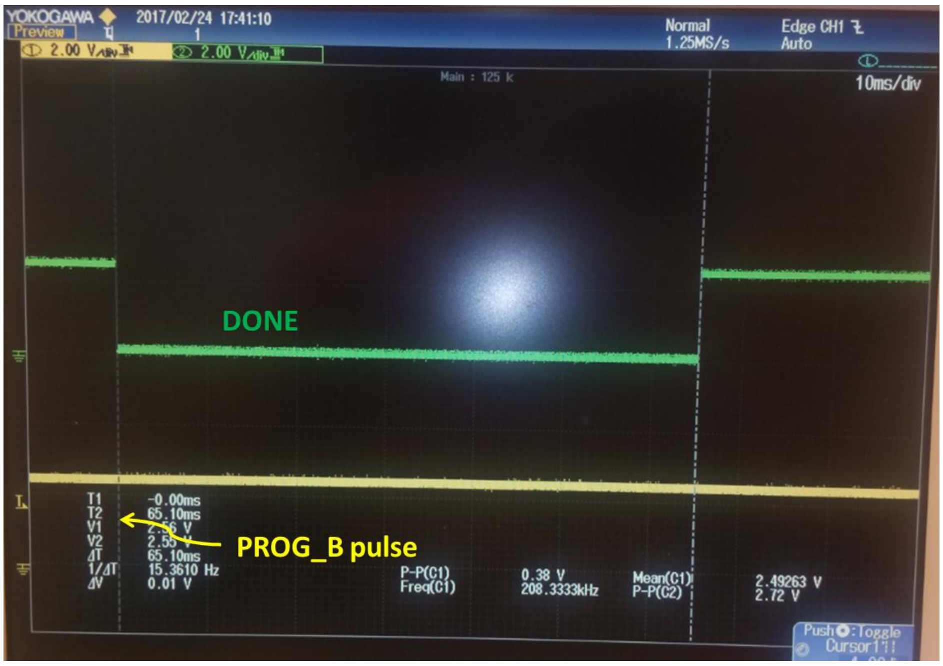


Figure 5: DONE signal low during Virtex-5 reconfiguration

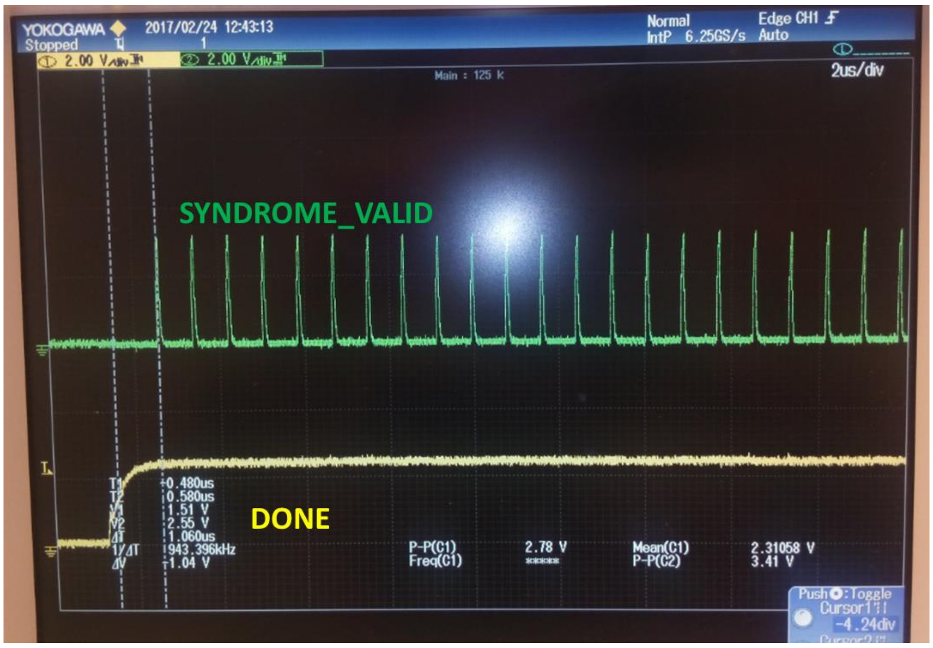


Figure 6: SYNDROME_VALID pulses after Virtex-5 reconfiguration

2.1.1.3 Command execution timing

We have measured the time that the DRM needs to process and execute each type of command message received from the LEON processor. This time is measured from the assertion of the interrupt signal that warns the DRM that there is a command message in the Mailbox DPRAM to be processed, until the DRM asserts the interrupt signal to warn the LEON that the answer message is ready in the Mailbox DPRAM. These times are shown in the next table.

Message	Execution time (ms)
Status	0.496
Perform Operation	0.505
Request Loaded Bitstreams	0.500
Store Partial Bitstream (1)	24.7
Store Partial Bitstream (2)	7.72
Store Partial Bitstream (3)	3.12
Load Partial Bitstream	17.45
Retrieve SEU statistics	0.503
Configure SEU mitigation	0.5

Table 2: Command execution times

Notes:

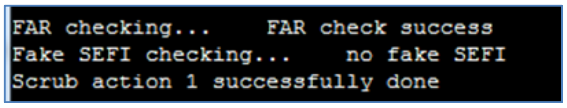
a) These times have been measured when the DRM runs an application that does not print any message out to the Tera Term console, as it would be the case in a real scenario.

b) When the DRM receives the Store Partial Bitstream command that includes the first fragment of a partial bitstream, firstly it has to erase the data flash memory area assign to that image. This fact explains that the execution of the first Store Partial Bitstream command takes more time than the next messages (time#1). The next Store Partial Bitstream command messages contain 2048 bytes of the new bitstream and the DRM has to write them into the data flash (time#2). Finally, the last fragment of the partial bitstream may have less than 2048 bytes, so the time that the DRM needs to execute the last Store Partial Bitstream command can be shorter (time#3).

2.1.2 Robustness and Self-Healing evaluation of the solution

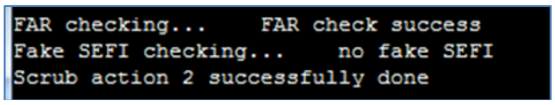
In order to evaluate the robustness and self-healing capability of the implemented solution including the DRM and the external control and monitoring agent (LEON processor and watchdog timer in the RTAX), we have induced errors in the Virtex-5 configuration memory and in the LMB BRAM memory of the DRM sub-system using the mechanism described in section 3.2 of D4.10. In this way we have simulated the different scenarios that could take place in a real situation under space radiation. In each scenario we have checked that the system can recover from the error. In most cases, the error does not have any impact on the DRM, so it can correct it and we can say that the Virtex-5 is self-healing. When the error does have a negative impact on the DRM design, the external monitoring agent comes into action and the system recovers after Virtex-5 reconfiguration. Here is a summary of the simulated scenarios and the results:

1. **Errors induced in the Virtex-5 configuration memory without negative impact on the DRM.**
   The following cases have been tested in this scenario:
   a) One bit is toggled in a configuration frame. The error is always detected and corrected. Just one correction action is done. The associated statistics counter (Single-Error Configuration Memory Frames counter) is increased in one unit and the next information message is printed out by the DRM:



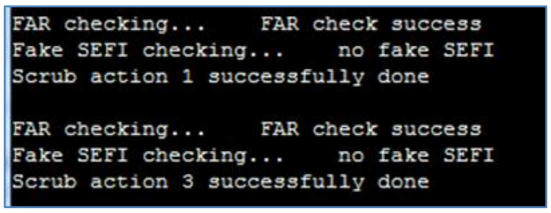
```
FAR checking...     FAR check success
Fake SEFI checking...     no fake SEFI
Scrub action 1 successfully done
```

   b) Two bits are toggled in a configuration frame. The errors are always detected and corrected. Just one correction action is done. The associated statistics counter (Double-Error Configuration Memory Frames counter) is increased in one unit and the next information message is printed out by the DRM:



```
FAR checking...     FAR check success
Fake SEFI checking...     no fake SEFI
Scrub action 2 successfully done
```

   c) Three bits are toggled in a configuration frame. The errors are always detected and corrected. In most cases two correction actions are done (scrub action 1 and scrub action 3), and in some cases just one correction action is done (scrub action 3). This happens because the ECC checking algorithm can infer that there is just one bit error. Since toggling this bit does not solve the problem, a global CRC error in the configuration memory is detected and a full device scrub action is done (this is action 3). One or two statistics counters (Single-Error Configuration Memory Frames counter and Full Scrubbing Processes counter) are increased in one unit. The next information messages (or just the second one) are printed out by the DRM:



```
FAR checking...     FAR check success
Fake SEFI checking...     no fake SEFI
Scrub action 1 successfully done

FAR checking...     FAR check success
Fake SEFI checking...     no fake SEFI
Scrub action 3 successfully done
```

   d) Four bits are toggled in a configuration frame. The errors are always detected and corrected. In the tests we have carried out just one correction action is done. This action can be scrub action 3 (which means full device scrubbing) or scrub action 2 (which means scrubbing of a frame with two bit errors; although 2 bit errors are reported instead of 4, all the bit errors are corrected because the configuration frame is scrubbed with data read from the data flash). It could happen that, as in the previous case, two correction actions (scrub action 1 and scrub action 3) were done, since it depends on the result of the ECC checking algorithm. In any case, the associated statistic counter(s) is/are increased in one unit and each information message printed out once.

2. **Errors induced in the Virtex-5 configuration memory causing design intrusive SEFI.**
These are errors that make the SYNDROME_VALID pulses stop. There could be no negative impact on the design functionality. However, new errors in the configuration memory wouldn't be detected. The only possible solution is FPGA reconfiguration. We have been able to simulate this situation and check that the watchdog timer in the RTAX detects that the SYNDROME_VALID pulses stop and asserts the PROG_B signal. For example, this happens when toggling the second bit of the configuration frame with address 0x001415. Figure 7 shows that if there are not SYNDROME_VALID pulses for 1.2 seconds, the watchdog timer asserts the PROG_B signal. After Virtex-5 reconfiguration the SYNDROME_VALID pulses resume.

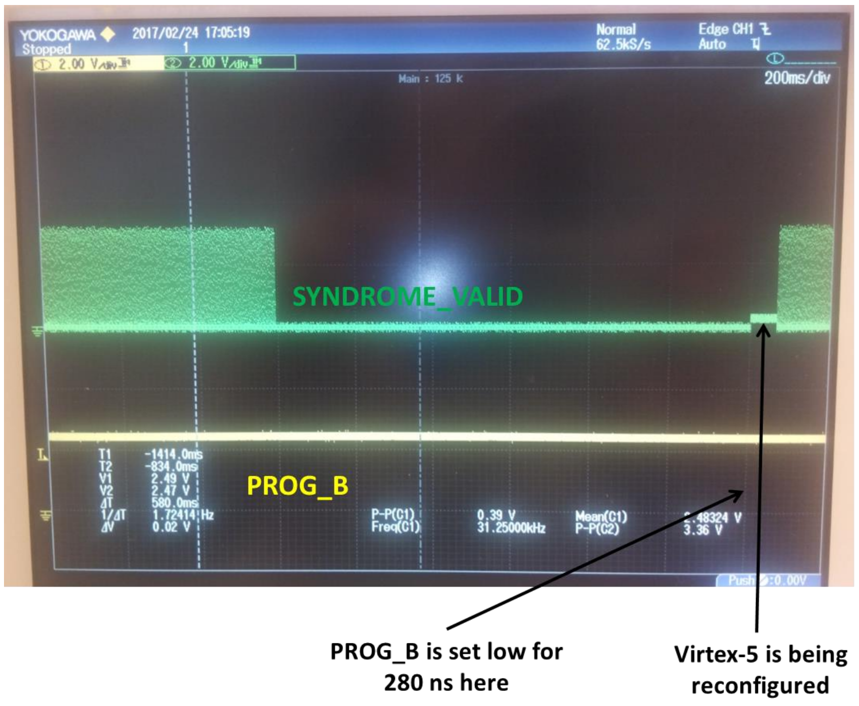


Figure 7: Recovering from design intrusive SEFI

3. **Errors induced in the Virtex-5 configuration memory causing visibility loss SEFI.**
These are errors that affect the ICAP interface. The design continues working but the DRM does not have visibility of the configuration memory and configuration registers through the ICAP port. In this situation, the DRM should set the SEFI_FLAG signal high and the external watchdog timer will assert the PROG_B signal low to reconfigure the Virtex-5 FPGA. We have been able to simulate this situation and check that the system behaves as expected and it recovers from the failure. For example, when toggling the 235th or 236th bit in the configuration frame with address 0x001A05, the error is detected by the Configuration Memory Monitor peripheral, but FAR register checking (which is done before any scrubbing attempt) fails. Consequently, as shown in xxx, the DRM asserts the SEFI_FLAG output, the external watchdog timer asserts the PROG_B signal low for at least 250 ns and the Virtex-5 is reconfigured.

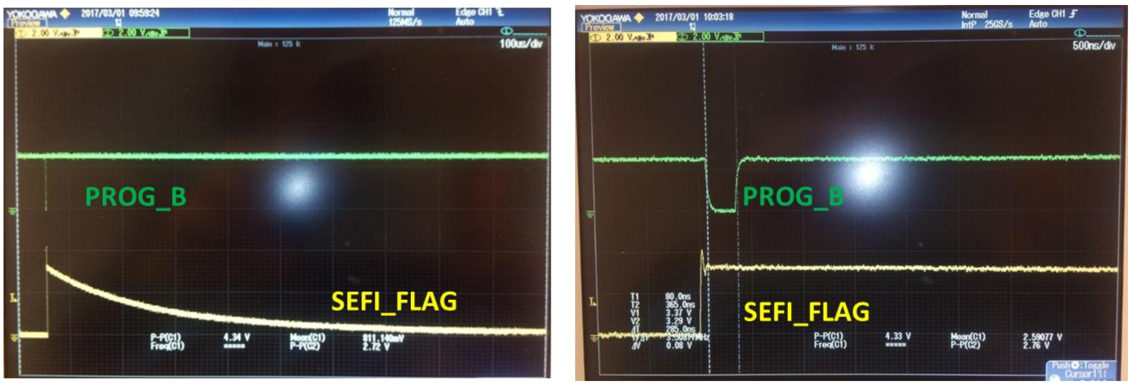


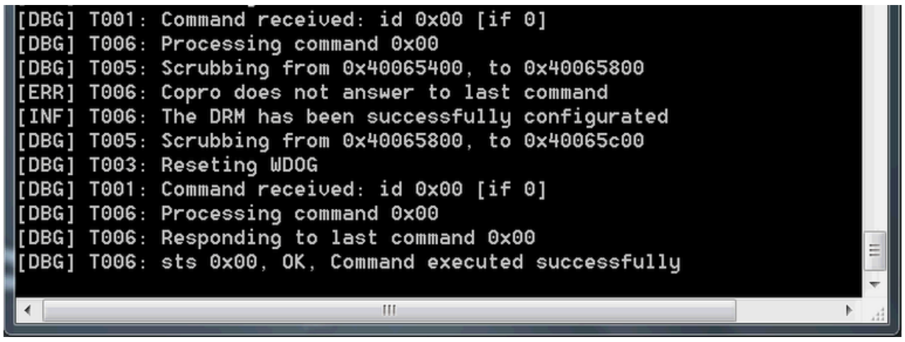
Figure 8: SEFI_FLAG assertion after unsuccessful FAR checking

4. **Errors that negatively impact on the DRM design.**
   Two possible situations are foreseen as indication that the DRM functionality has been affected:
   a) The DRM does not answer to the command requests received from the LEON processor.
   b) The statistics counters are increased at a high rate. This means that errors are detected in the configuration memory, but the DRM cannot correct them.

   When the LEON detects any of these situations, it requests the watchdog timer to reprogram the Virtex-5 FPGA.

   We have been able to simulate this scenario by, for example, toggling the 3rd and 4th bits at MicroBlaze LMB memory address 0x00000014. After injecting these errors, we request[1] the LEON processor to send the *Status* command (command id = 0x00). In the system terminal, the LEON prints out the messages shown in Figure 9. Since the DRM does not answer to the command request ("Copro does not answer to last command"), the LEON timeout expires and it requests the watchdog timer to reconfigure the Virtex-5 device ("The DRM has been successfully configured"). After reconfiguration, we request the LEON to send the *Status* command and this time it receives the answer from the DRM.

   (1) Note: in a real scenario, the LEON processor will periodically send the *Status* and *Retrieve SEU Statistics* commands. During the validation tests, these commands are sent under user's request for debug purposes.



```
[DBG] T001: Command received: id 0x00 [if 0]
[DBG] T006: Processing command 0x00
[DBG] T005: Scrubbing from 0x40065400, to 0x40065800
[ERR] T006: Copro does not answer to last command
[INF] T006: The DRM has been successfully configurated
[DBG] T005: Scrubbing from 0x40065800, to 0x40065c00
[DBG] T003: Reseting WDOG
[DBG] T001: Command received: id 0x00 [if 0]
[DBG] T006: Processing command 0x00
[DBG] T006: Responding to last command 0x00
[DBG] T006: sts 0x00, OK, Command executed successfully
```

Figure 9: LEON messages when the DRM does not answer to the commands

After all these tests, we can conclude that:

The implemented Reliable and Self-Healing Dynamic Reconfiguration Manager is a robust solution so that Xilinx FPGA partial reconfiguration can be safely used to dynamically change, add or remove peripherals in mixed-criticality systems, in general, and in the space environment in particular.

2.2 Performance prediction, Architecture Test and Benchmarking

The ability to properly reuse scarce resources on a spacecraft can determine the success or failure of a mission. Therefore, increasing performance while maintaining or reducing costs is required to allow the full potential of space to be exploited. For such purpose, it is required to adapt the hardware advances developed for other activities to be adopted in the space domain.

In this context, multiprocessor systems-on-chips (MPSoC) are one of the key features of current technology. However, its integration in the space business is still low, due to two main reasons. On the one hand, while typical MPSoCs on complex microprocessors, tools oriented to develop space systems need to consider microcontrollers as the main processing elements, instead of microprocessors. Then, it is required to transform previous MPSoC concepts into Multi Controller System on Chip (MCSoC).

On the other hand, space applications have to handle strict non-functional requirements such as criticality, safety, timeliness, security and reliability. Thus, the development of these systems is typically long and complex. However, there is a lack of methodologies and tools to support the exploitation of these new technologies in the scope of systems considering the peculiarities of space applications. As a result, it is important to develop tools capable of ensuring that the design process is in the right direction from the very beginning, since going back in the designs is very costly.

Several alternatives have been proposed to simulate processor operation in order to create virtual platforms. These alternatives propose working at different levels of abstraction, providing different tradeoffs in terms of accuracy vs. performance and usability. Among them, host-compiled simulation is one of the most promising solutions since it provides best performance and it is easy to use. This technique is based on static annotation of the original SW source code with the expected performance it would have in the target platform. After that, compilation and execution of the annotated code in the host computer is done.

Host-compiled simulation is a technology oriented to generate fast virtual platform models, where the different design alternatives can be checked at the beginning of the design process. Fast virtual platform models are quite important in embedded system development since they enable embedded software development, performance analysis and verification by running code on a prototype of the hardware platform much earlier than the real platform and the final SW adaptations to the specific HW are available.

In that way, during this project, a previous tool (VIPPE) has been expanded and adapted to be able to model these multi-controller systems, especially LEON3-based systems. VIPPE is a tool oriented to create virtual platforms based on host-compiled simulation. It can model multi-core platforms, including SystemC peripherals, and it integrates an abstract model of the Operating System, providing several APIs, such as POSIX, which enables to simulate embedded Linux platforms.

For example, it is currently perfectly possible to run a set of applications on a FPGA programmed to implement a platform containing one or two LEON processors. However, it is not easy to evaluate the real performance and the side effects resulting from the interactions of these applications since the frequency of the FPGA, and thus, the execution speed, is much smaller than in the final chip.

This approach has the capability to approximate the resulting simulation speed to the speed of the native execution of the original source code in the host computer. Furthermore, it does not require porting the SW (such as the compiler or device drivers) to the target platform, which is extremely useful specially for platforms that are initially developed on FPGAs, where multiple HW configurations, including application-specific HW can be design. In this context, as the proposed simulation tool has the capability of predicting the performance and behavior of the different applications on the final chip, it can help designers to check if the HW alternatives selected adequate or not, reducing the number of HW redesigns required. Additionally, the tool enables to optimize the application SW considering the details of the HW platform under development before a real prototype is available.

To demonstrate that capabilities and evaluate its correct behavior, in this task of the project, the tool has been used to explore the effect that different SW optimizations will have in the target board.

2.2.1 Performance prediction

The work performed in this task has focused on the evaluation of the impact that different SW optimizations preformed in sequential C codes will have in a certain HW platform, specially when transforming sequential codes into concurrent codes capable of take advantage of multi-core platforms. During these experiments, the target platform modeled has been a mono and dual-core LEON3-based platform running on a XILINX ML506 FGPA-based platform.
The fact that an FPGA-based platform has been selected could be weird since one of the main goals of the tool is to model the behavior of platforms before its physical availability. However, this board has been used in order to enable the comparison between the performance results obtained by the simulator and the results obtained from real executions. Considering that the number of clock cycles required to by the HW does not typically change when selecting a different technology, once the simulator has been validated against an FPGA-based implementation, the extrapolation of the results to other kind of implementations of the same VHDL code is simple. It can be easily done using the simulator, just by adapting the clock frequency, and other numbers, such as the delay of the external SDRAM chips.
To check the tool, two SW applications have been selected: a CCSDS 122 coder, and a AES coder. Both applications have been used to evaluate how SW parallelization behave in a multi-core platform. For such purpose, first they have been evaluated running the initial, sequential version on a mono-core platform, under a Linuxbuild OS. Then, the simulator has evaluated the impact of the modifications done in the SW code to enable concurrency, and the effect of using a dual-core platform. The expected behavior of other configurations with more cores have also been simulated, but not verified in the board, since its FGPA does not enable the implementation of more cores.

Development of the UML Models for the examples

VIPPE tool uses standard UML/MARTE models as inputs. In these models, the user describes the SW components of the application and their interconnections, the HW platform, including all the parameters required to perform a proper simulation, and the compilation information required to create the target executables. Since the application SW in the examples used has been developed as a single component, the UML models mainly focus on the description of the platform: processor (type, frequency, …), caches (size, sets, …), bus, main memory, etc.

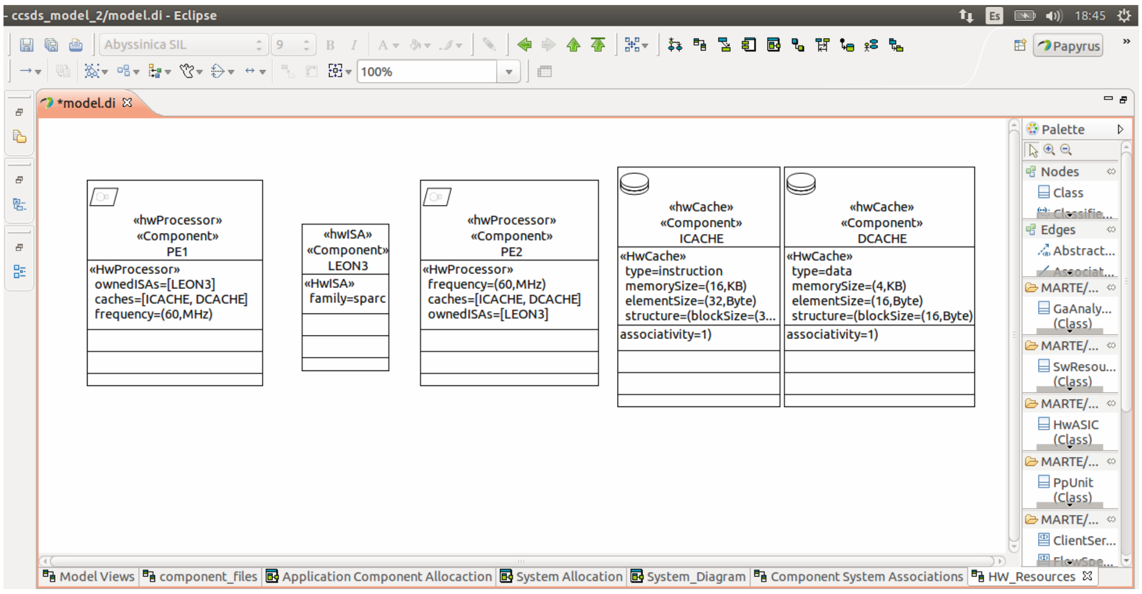

ccsds_model_2/model.di - Eclipse
*model.di
«hwProcessor» «Component» PE1
«HwProcessor» ownedISAs=[LEON3] caches=[ICACHE, DCACHE] frequency=(60,MHz)
«hwISA» «Component» LEON3
«HwISA» family=sparc
«hwProcessor» «Component» PE2
«HwProcessor» frequency=(60,MHz) caches=[ICACHE, DCACHE] ownedISAs=[LEON3]
«hwCache» «Component» ICACHE
«HwCache» type=instruction memorySize=(16,KB) elementSize=(32,Byte) structure=(blockSize=(3... associativity=1)
«hwCache» «Component» DCACHE
«HwCache» type=data memorySize=(4,KB) elementSize=(16,Byte) structure=(blockSize=(16,Byte) associativity=1)
Palette
Model Views | component_files | Application Component Allocation | System Allocation | System_Diagram | Component System Associations | HW_Resources

Figure 10: detail of the HW components used to model the platform

Then, the model can be used to automatically generate the executables required both for simulation in the PC and execution on the target board, which is important to speed-up the comparison processes. Additionally, just changing the parameters in the model it is possible to use the simulator developed to evaluate multiple configurations without requiring its implementation in the target board.

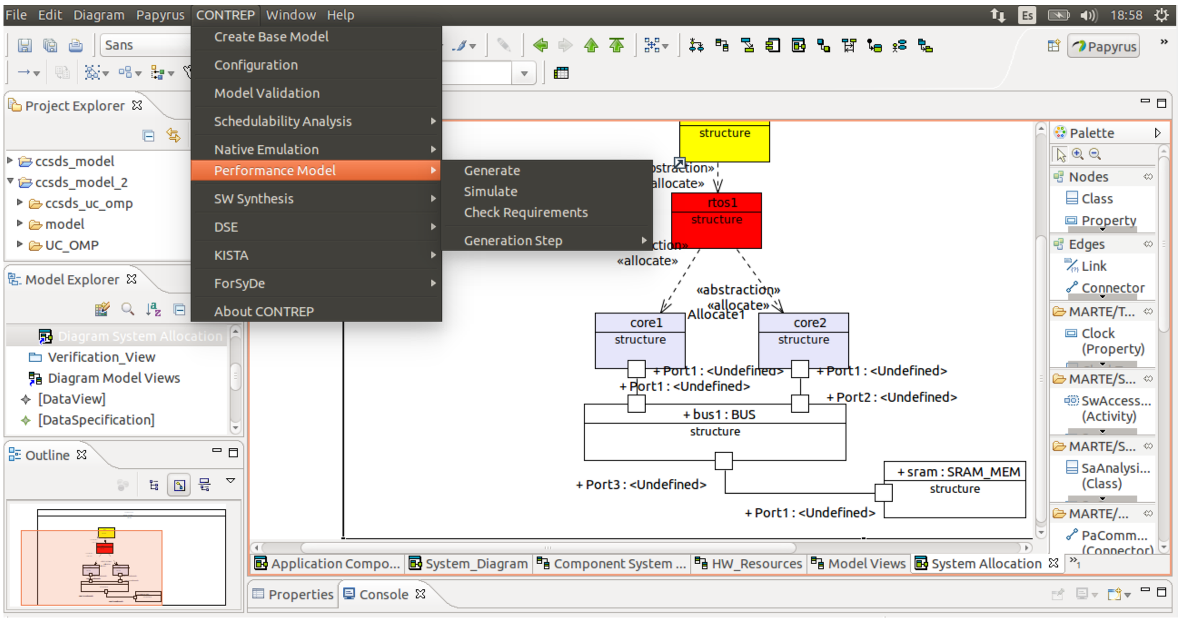


Figure 11: Detail of the SW allocation in a HW platform containing two processors.

2.2.1.1 CCSDS 122 use case

To evaluate the accuracy of the tool developed, the first experiment performed was the execution of a CCSDS 122 coder both in the simulator and the real platform. This enables comparing the results obtained with the simulator developed in the project with respect to results obtained from the real implementation. For such purpose, a XILINX ML506 platform (Virtex 5) board has been used. This board has been configure integrating platforms with 1 and 2 LEON3 cores and a “FPU-lite” unit has been added to perform floating point instructions. Regarding the SW platform, both comparisons in bare metal, and with Linuxbuild OS have been performed.

Since the LLVM has a port for “sparc v8” but not specifically for LEON3, some problems with the assembling step have appeared when generating executables for the target board. To solve them, the assembly code obtained with LLVM was transformed into binary with the corresponding GNU tool (“as”) instead of with the LLVM tool.

To obtain information about number of cycles and instructions in the real platform to perform the comparisons, the peripheral “l3stat” provided in the standard distribution of LEON3 VHDL codes (version 1.4 onwards) has been used. Accessing “l3stat” registers for starting/stopping the counters adds some overhead, which has been measured using an empty “main” function. Then, this value has been subtracted from the real measures before putting them in the tables below. In the case of 2 cores with OS, the overhead is not constant, and thus values measured has some small uncertainty. This uncertainty can be estimated comparing the execution time of both cores, which should be the same. For the CCSDS example, estimation of execution times with input images of size 1000x1000 pixels have been compared running under Linux OS, with a real platform containing 1 and 2 cores.

For 1 core, results are:

	Board	VIPPE	Error %
Intructions	396363906	392061803	1,1
Cycles	792159757	724561621	8,5
Time(ms)	13202,6	12407,5	6,0

When moving to a dual-core platform, the mapping of tasks to cores change on each execution, so instructions and cycles on each processor cannot be compared separately. In this case, comparison is only possible analyzing the overall execution time.

Considering that issues, the results obtained are:

1 thread

Board	CPU0	CPU1	VIPPE	CPU 0	CPU 1	Error %
Instructions	387605066	39716098	Instructions	413141813	32904	3,3
Time(ms)	13786,4	13822,4	Time(ms)	12737,65		7,8

2 threads

Board	CPU0	CPU1	VIPPE	CPU 0	CPU 1	
Instructions	191801968	232804017	Instructions	214835297	198339547	2,7
Time(ms)	9064,5	9104,6	Time(ms)	7399,92		18,7

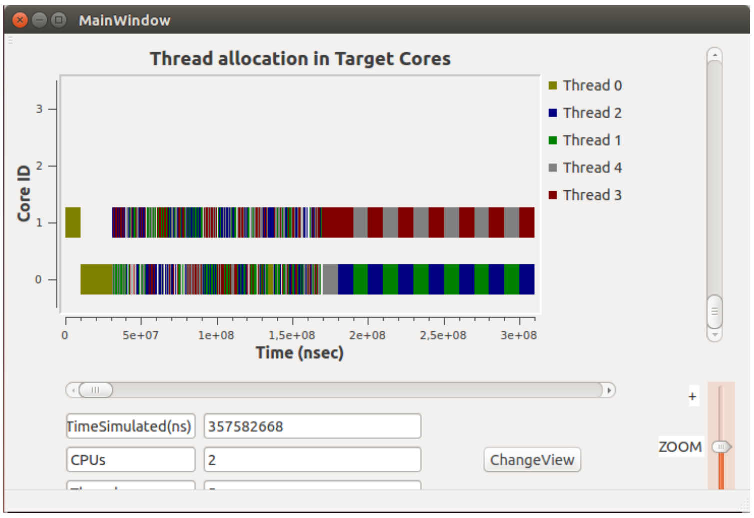
As can be seen, the parallelization integrated in the code is capable of reducing the execution time taking advantage of the dual core. As the table shows, when considering the example, the accuracy of the estimation tool is still quite good since all errors are below 20%.

Regarding simulation speed, the results are the following:

Time to execute in the board: 13seg
Time required by VIPPE simulator: 0.5 seg

Thus, it demonstrates that the developed tool is an interesting alternative for early evaluation of multiple designs, especially if we consider the time required to synthesize a full HW platform in order to be evaluated in an FGPA as the one used in the experiments.
The tables also show that the modeling of the dual-core platform results in an increase of the percentage of error, not in number of instructions but in number of cycles. After some analysis, there are two main reasons for that divergence. The first one is that the OS model of the simulator cannot exactly emulate the task allocation performed by the real OS. Therefore, different allocations have impact on cache operation and result in different execution times, while maintaining the number of instructions executed.
The problem here, is that there is no a fixed order in the real execution. Each execution of the same application in the real board result in a different ordering, with different execution times. Thus, although the error can be minimized, it is not possible to fully eliminate it, since there is not a single value for the comparison.
The second problem is due to conflicts in the bus. Typically, when several processors share the bus, there is a distribution on the bus conflicts. However, due to the independency on the data, and since the parallelization has been implemented in a fork/join sequence, both processors try to execute the same instructions at the same time, resulting in a constant collision of load/store instructions. As a result, the example becomes a sort of corner case in terms of bus collision estimation.

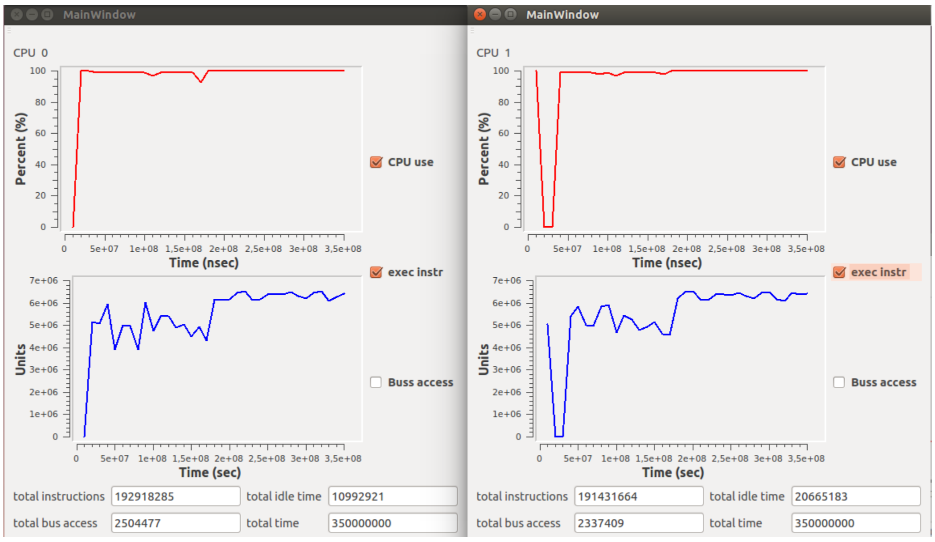
Together with performance estimation, the developed tool also enables the analysis of how the different resources operate during code execution. For that purpose, VIPPE simulator provides a graphic interface where the information is displayed. For example, next figure shows how the different threads are executed in the two-core platform:



TimeSimulated(ns) 357582668
CPUs 2

Figure 12: Execution of the different threads of the CCSDS on the two-core platform

At the same time, it is possible to evaluate the degree of utilization of the processors:

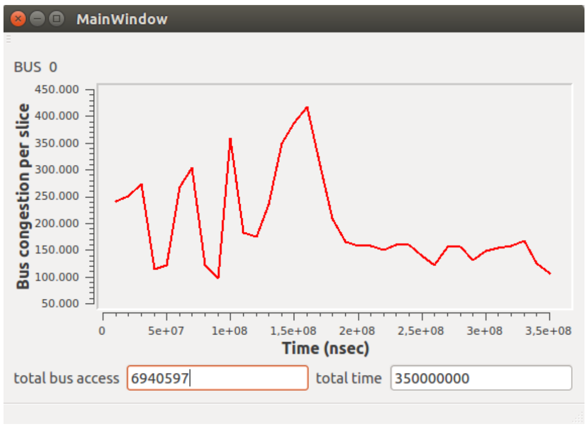


total instructions	192918285	total idle time	10992921
total bus access	2504477	total time	350000000

total instructions	191431664	total idle time	20665183
total bus access	2337409	total time	350000000

Figure 13: CPU utilization and number of instructions executed.

As shown in this example, most of the time, the two processors are busy, executing code, however, the number of instructions executed are not always the same. The first half of the coding has a lower value than the second part. The reason is the stalls resulting from accesses to the bus. It can be seen in the CPU utilization and number of instructions executed.



total bus access	6940597	total time	350000000

Figure 14: Bus transfers in the CCSDS example.

2.2.1.2 AES-128/256 use case

As a second example, and AES coder has been applied. Since AES typically operates with block sizes of 128 bits, input data can be divided into sets of 4 words for the computations. With this division, there are no dependences among the sets, and each set of 4 words can be operated independently from the rest of the sets, enabling full concurrency.
Taking advantage of that, the original AES code taken as golden model, has been modified, and parallelized supporting a variable number of threads (1,2,4, etc.). Taking the original code and the parallelized codes, first step done was to simulate and execute in the real board a platform containing one LEON3 core.
Results obtained are the following.

Sequential Bare C 1 Core

		Board	Simulator	Error %
	Instructions	9,21E+08	9,61E+08	4,3
	Cycles	1,59E+09	1,49E+09	6,1
	Time (s)	26,4	24,8	6,1

Concurrent Linux 1 Core

		Board	Simulator	Error
1 Thread	Instructions	1,06E+09	1,12E+09	5,9
	Cycles	2,02E+09	2,04E+09	0,6
	Time (s)	33,7	33,9	0,6
2 Threads	Instructions	1,07E+09	1,12E+09	5,2
	Cycles	2,05E+09	2,01E+09	2,0
	Time (s)	34,2	33,3	2,3
4 Threads	Instructions	1,07E+09	1,12E+09	5,5
	Cycles	2,05E+09	1,98E+09	3,4
	Time (s)	34,2	33,0	3,4

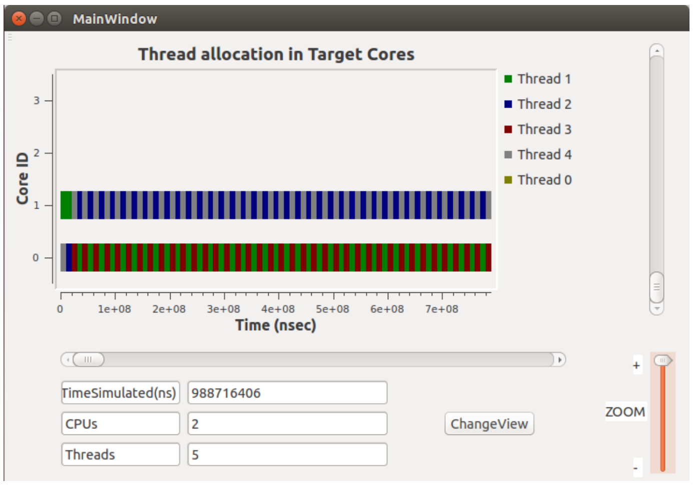
As it can be seen, accuracy on the results obtained for these experiments is quite high.
A second experiment done with the example was to apply a dual-core LEON3 board, to the parallelized application, using with 1, 2 and 4 threads to execute the application:

Concurrent Linux 2 Cores

			Board	Simulator	Error %
1 Thread	Core 0	Instructions	2,60E+07	1,13E+09	
	Core 1	Instructions	1,06E+09	723	
	Total	Instructions	1,09E+09	1,13E+09	4,3
	Core 0	Cycles	2,30E+09	2,19E+09	5,1
		Time (s)	38,375	36,446	5,0
2 Threads	Core 0	Instructions	5,33E+08	5,66E+08	
	Core 1	Instructions	5,37E+08	5,66E+08	
	Total	Instructions	1,07E+09	1,13E+09	6,1
	Core 0	Cycles	1,22E+09	1,09E+09	10,9
		Time (s)	20,382	18,18	10,8
4 Threads	Core 0	Instructions	5,38E+08	5,71E+08	
	Core 1	Instructions	5,37E+08	5,62E+08	
	Total	Instructions	1,08E+09	1,13E+09	5,4
	Core 0	Cycles	1,24E+09	1,02E+09	17,8
		Time (s)	20,609	16,961	17,7

Tables show that the error is similar than in previous example, and it comes from the same reasons. In that context, it can be concluded that a deeper analysis of bus collisions can be required to minimize estimation error in case of multithread systems
Nevertheless, the final error is below 20%, which is a good result for an early estimation. It is important when compared with the simulation time achieved. While the real execution of the code takes from 17 to 36 seconds in executing the codification of the proposed sequence, VIPPE simulator takes 2.5 to 3 seconds in performing the simulation. This means that it is a very good solution to evaluate large SW applications, since they take an order of magnitude less that the execution in the real FGPA-based board.

Finally, the graphic interface can be also used to see how the parallelization of the AES result in a more constant operation:



TimeSimulated(ns) 988716406
CPUs 2
Threads 5

Figure 15: Execution of the different threads of the AES example on the two-core platform

2.3 Distributed platform TTEthernet configuration tooling

In the last phase of EMC2, further research work has been performed on the TTEthernet toolchain in order to support the network configuration for the network nodes targeted for space platform integration. Such nodes can reside within the same multicore chip, or between several multicore chips within one control computer, or between several control computers within the satellite. Whereas the previous phase was concentrated on the development of the network planning and binary generation, the last phase has particularly targeted the creation, visualization and subsequent refinement of individual time-critical traffic flows over TTEthernet networks.

As described in D9.5, the TTEthernet toolchain allows for the planning of the TTEthernet network with its networking nodes, the creation of physical and logical channels in the network and mapping the corresponding virtual links to the devices connected in a so-called “network description” (ND). The tool performs checking and validation of the ND and checks if the critical traffic can be scheduled according to its timing constraints (i.e. period, latency, bandwidth allocation). Based on this calculation, the schedule for time-triggered traffic is provided and corresponding Network Configuration files (NC database) are created which provide the individual configuration files for each of the devices in the network. New to this environment is the Device Configuration Editor.

To allow configuring the TTEthernet device configurations a GUI based editor enabling the user to edit the device configurations in a convenient way has been developed. This editor allows modifying the mapping between the partitions/tasks of the applications/host with the partitions and data-ports of the TTEthernet End System to be configured without using the network/system level tools. Moreover it checks the configuration for consistency.

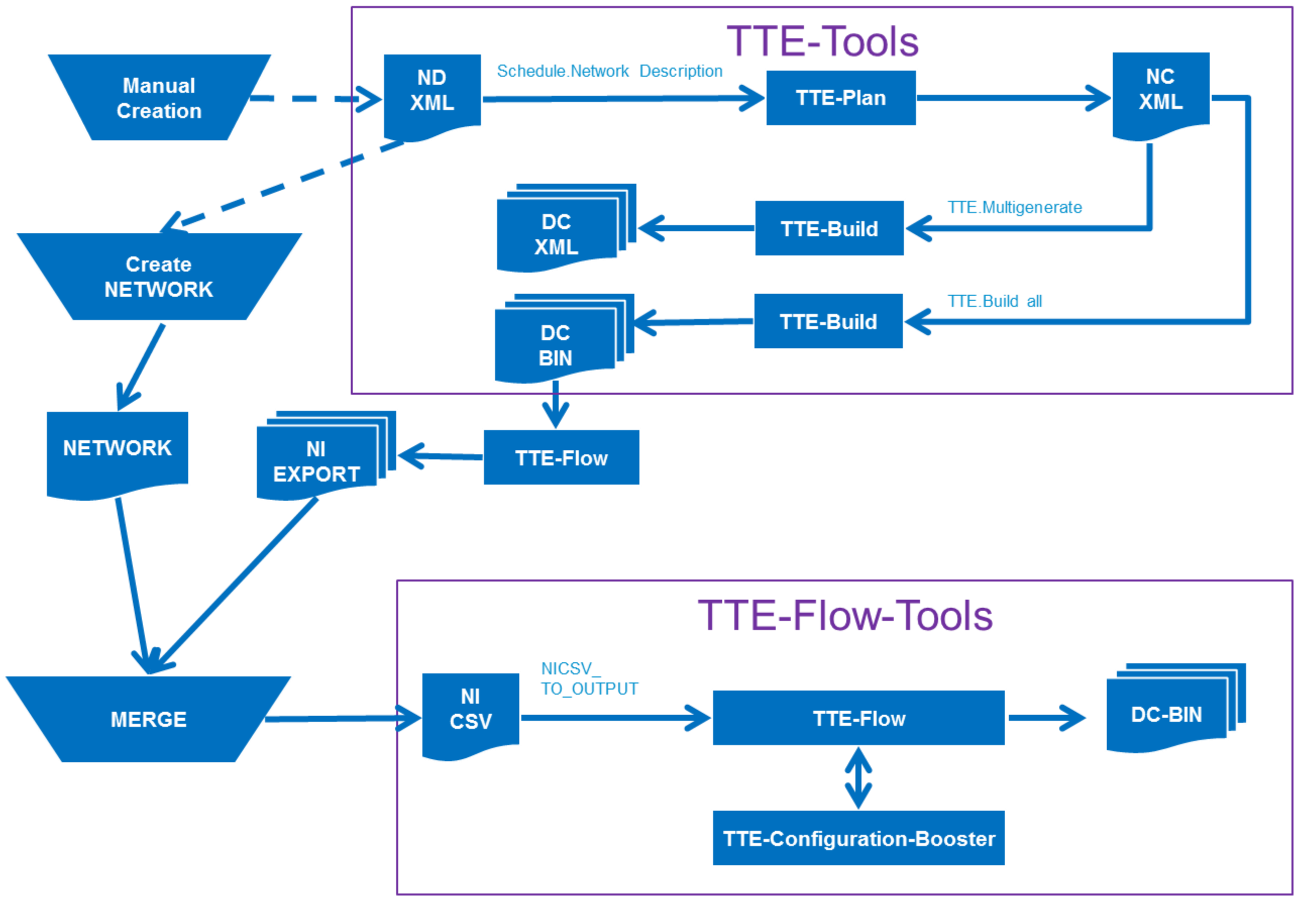


Figure 16: Device Editor and TTE-Tools interaction

Figure 16 illustrates the device editor toolchain and also provides the interface description to the TTE-Tools. As illustrated, the TTE-Tools configuration can be loaded back in to the device editor tool called TTE-Flow via an additional format called Network Information (NI). This format allows loading one or multiple device configurations into the TTE-Flow tools. The NI format is used to edit the devices in GUI mode.

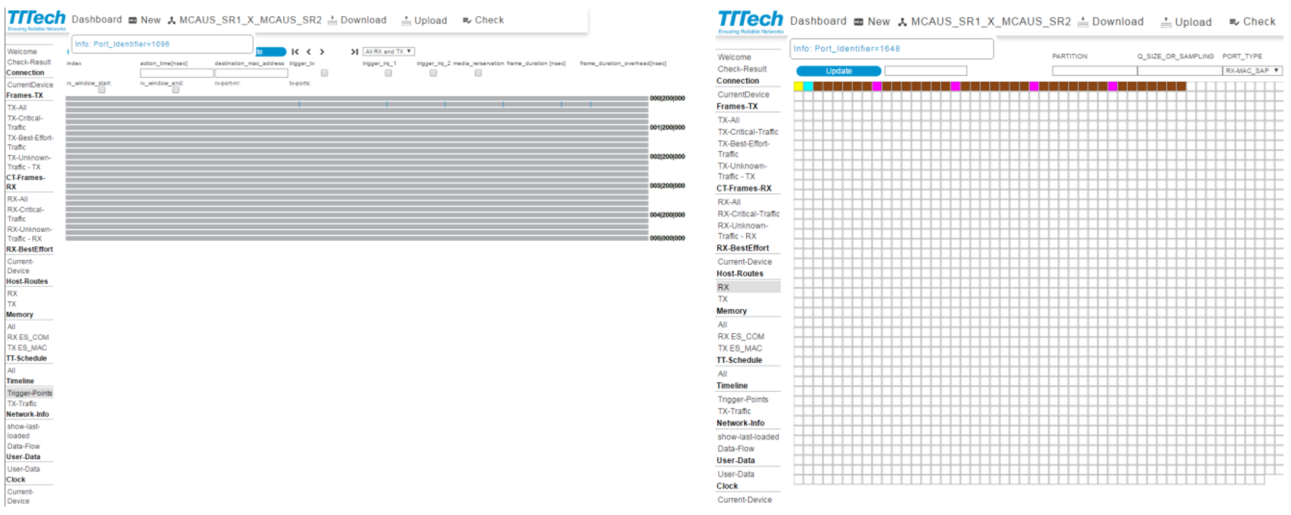


Figure 17: Device Editor and TTE-Tools interaction

The GUI is based on HTML5 as illustrated in Figure 17 and therefore shows a nice visual graphic interface for the user. It further allows abstracting the complexity of the device configuration by providing graphical illustrations.

2.4 Heterogeneous TTNoC Architecture

The main focus of TUW in the last reporting period were integration capabilities of the architecture described in D4.3, D4.4, D4.5, D9.1 and D9.3. It is heterogeneous many-core architecture implemented on top of a hybrid SoC platform. In particular, for the purposes of the project we used an Arria V SoC development board (see Figure 18). Three basic components of a hybrid SoC platform are: hard processing subsystem, FPGA and interconnect. We identified a number of ways how to utilize advantages of such architecture in mixed-criticality environments.

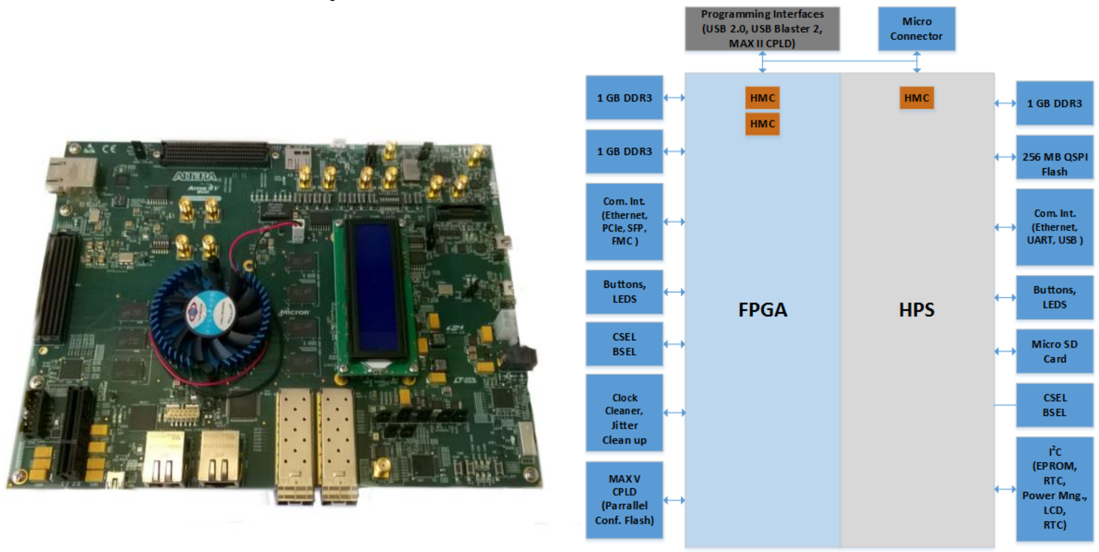

Programming Interfaces (USB 2.0, USB Blaster 2, MAX II CPLD)
Micro Connector
1 GB DDR3
1 GB DDR3
Com. Int. (Ethernet, PCIe, SFP, FMC)
Buttons, LEDS
CSEL BSEL
Clock Cleaner, Jitter Clean up
MAX V CPLD (Parrallel Conf. Flash)
HMC
HMC
FPGA
HMC
HPS
1 GB DDR3
256 MB QSPI Flash
Com. Int. (Ethernet, UART, USB)
Buttons, LEDS
Micro SD Card
CSEL BSEL
I²C (EPROM, RTC, Power Mng., LCD, RTC)

Figure 18: Arria V SoC Development Board and Block Diagram of the Architecture

The main focus in the last period was to finalize the architecture and test basic functionality. We experimented with tool integration process, fault tolerance, scalability, and modular design.

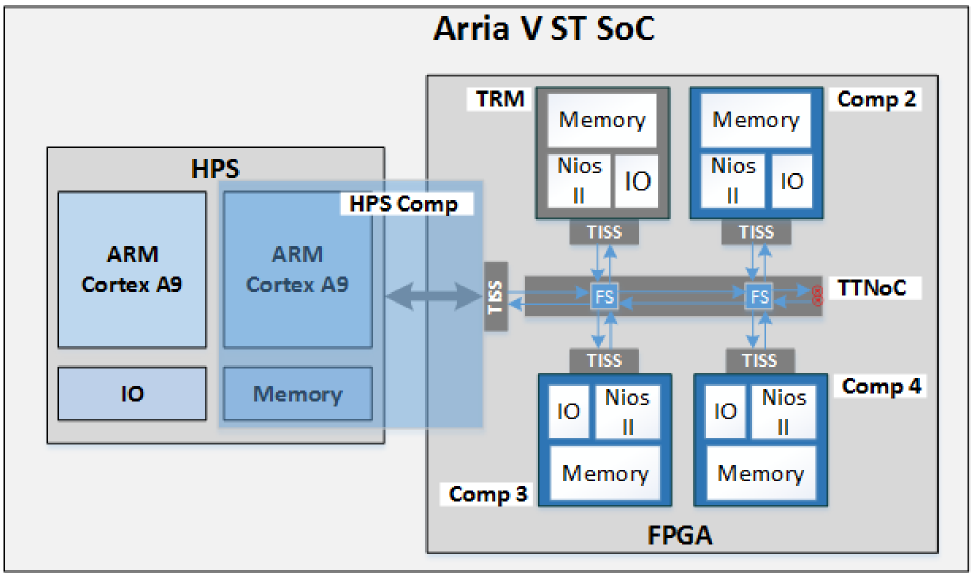


Figure 19: Heterogeneous TT MPSoC architecture

An arbitrary version of the architecture includes five many-core components designated as µComponents, four of which are based on Nios 2 processing units and single ARM component that occupies both HPS and FPGA. Figure 19 provides an overview of the current layout and its integral parts. Each component was built using generic template which can be configured according to needs of an application.

A major challenge for such architecture and its usability is a tool integration and ability to automate design process. One of the goals in the project was to explore this topic and identify the possibilities for an automated tools that integrates whole process from hardware design to the application. This would include:

- A component design, with implementation of the processing unit and its properties (e.g., type of the core, frequency, interrupts, memory management etc.), memory hierarchy and peripherals.
- Further, it considers TTNoC configuration, routing tables, global time settings and external synchronization.
- Application mapping and communication schedules.

Other important aspect of our work was on fault tolerance capabilities of the platform and the synergy with TTNoC architecture. Some of the important aspects are reconfiguration capabilities on the platform level. The ability to reprogram or reconfigure HPS and FPGA in case of fault or an error occurrence.

The simple architecture presented above is uses very few resources on the FPGA. The implementation presented in Figure 19 uses about 12% of the FPGA logic elements, 22% of block memory bits and 12% of pins. This platform is capable of hosting a much larger architecture and also the architecture can be implemented on a cheaper lower end platform as well. However, it is fair to say that in order to establish higher performance capabilities the components must be equipped with larger memory and higher frequency processing units.

In our evaluation of applications for space above mentioned properties could be highly beneficial. Especially the fault tolerance aspect of the work and the heterogeneous design principle of the design, as additional more dedicated components could be integrated in addition. The work related to the architecture has been also published in (Haris Isakovic, 2016).

3. MPSoC Software and Tools for Space

A space domain application based on a real example has been designed and evaluated with the tool suite art2kitekt. Both software and hardware elements have been modeled and analized with the tool. In tight collaboration with TASE, the tool suite art2kitekt has been developed and adapted to guide the engineer through the steps of system design and analysis for a space domain task.

Following an agile methodology for the tool suite art2kitekt software development, since October of 2016, which was the deadline for deliverable D9.5, a new internal release has been reached. Next the main achieved targets are listed:

- 17.04 – 9th internal "release:
  - Stage System Analysis: **Sensitivity analysis has been** integrated with RTA to provide useful information for changing the system model, by giving a measure of those computation times that must be reduced to achieve feasibility, those that can be increased, or by providing the range of feasible periods for selecting the proper task activation rates.
  - Stage Code Generation: New functionality added to automatically generate source code in C language for **RTEMS platform support**.
  - A new stage named **"Deployment Model"** has been designed, implemented and integrated into art2kitekt to better organize the input data and the results provided by the tool suite.

Task and flow allocation to available hardware can be now perfectly separated from the platform and application models.

Nevertheless, development of the tool suite art2kitekt has not finished and other interesting features will be added in the near future:

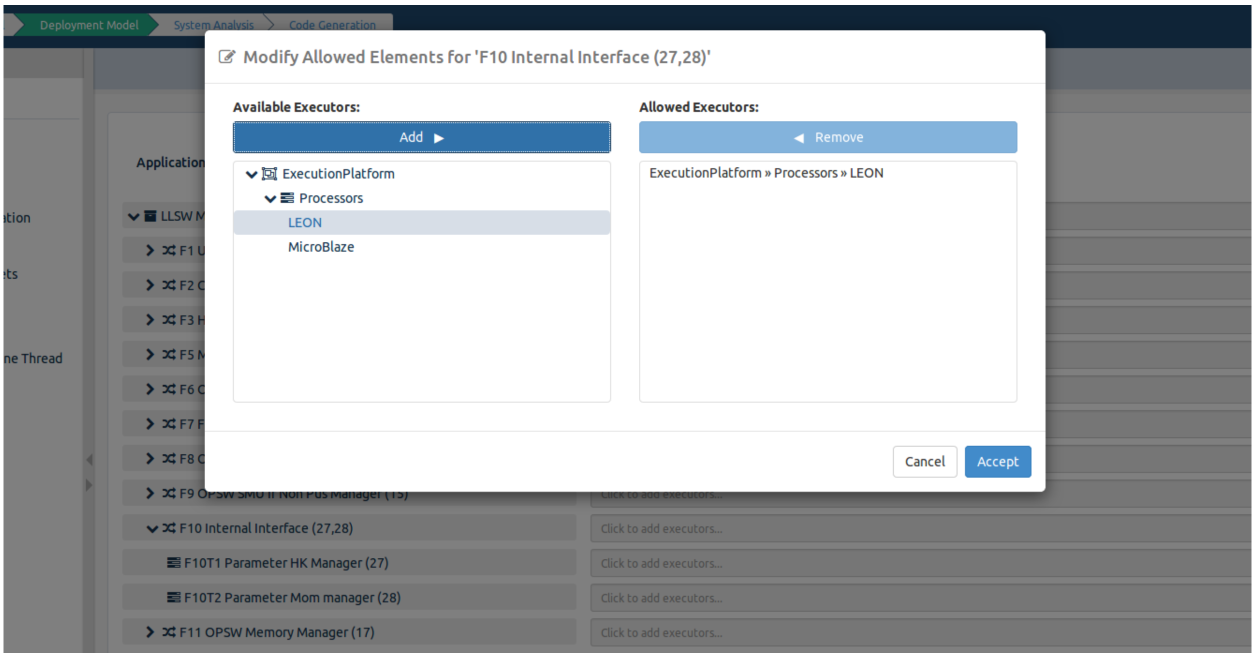
- Next 17.xx releases:
  - Stage Application Model: Support in the data model for **mixed criticality** tasks.
  - Stage System Analysis: An **RTA improvement** to tackle with "mixed criticality" tasks.
  - Stage System Analysis: New feature in the RTA to enable **task assignment to different processors**.

3.1 Application of EMC2 technical developments

From WP02 "Executable Application Models and Design Tools for Mixed-Critical, Multi-Core Embedded Systems", and specifically from Task 2.4 "Code generation and offline analysis tools", the tool suite art2kitekt has been further developed and adapted with UC 9.2 and thus to space domain requirements. Once more, valuable feedback has been provided to the art2kitekt tool suite development in different aspects as: analysis algorithms requirements, usability tips, source code examples to automate code generation.

A multi core platform from the space domain context has been modeled for evaluation purposes. Then, an application composed by an example task set has been also modeled. After that, with the RTA algorithm offered by the system analysis stage, tasks and flows have been mapped to the corresponding cores of the execution platform in a way that system feasibility is guaranteed. Finally, with the code generation stage, a low-level skeleton of the source code is automatically generated and prepared to run in a RTEMS operating system.

Thanks to the new stage known as "Deployment Model", flows can be manually assigned to available hardware resources. For example, in the next figure it can be seen that the flow "F10 internal interface" composed of two tasks has been modeled to be executed into the "LEON" core.



Modify Allowed Elements for 'F10 Internal Interface (27,28)'

Available Executors:

- ExecutionPlatform
  - Processors
    - LEON
    - MicroBlaze

Allowed Executors:

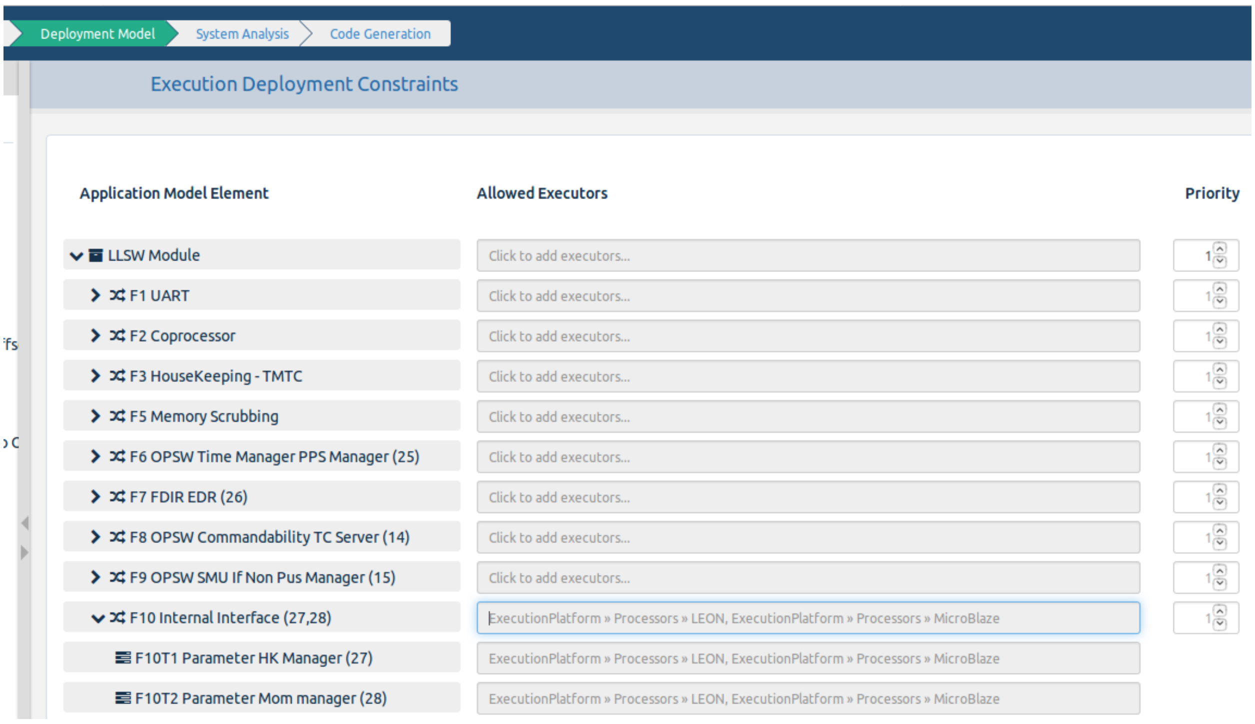
ExecutionPlatform » Processors » LEON

Application

- LLSW M
  - F1 U
  - F2 C
  - F3 H
  - F5 M
  - F6 C
  - F7 F
  - F8 C
  - F9 OPSW SMU II Non Pus Manager (15)
  - F10 Internal Interface (27,28)
    - F10T1 Parameter HK Manager (27) — Click to add executors...
    - F10T2 Parameter Mom manager (28) — Click to add executors...
  - F11 OPSW Memory Manager (17) — Click to add executors...

Figure 20: Stage Deployment model - Allowed elements

This information is codified into the data model with a new entity which allows decoupling platform hardware from application software at the modeling phase.

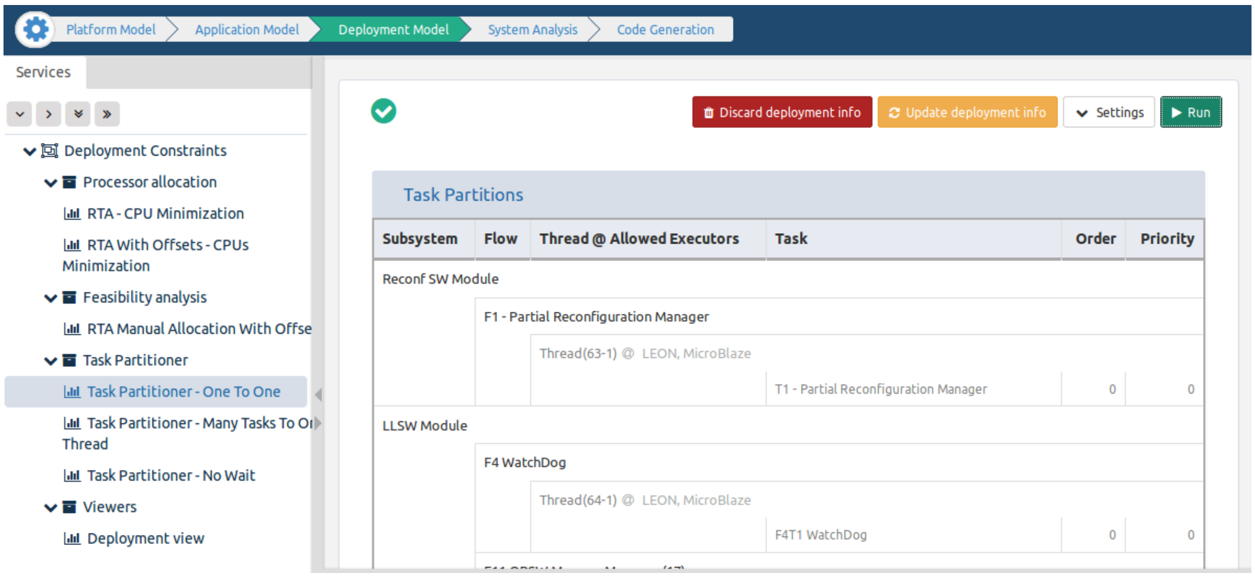


Execution Deployment Constraints

Application Model Element	Allowed Executors	Priority
LLSW Module	Click to add executors...	1
F1 UART	Click to add executors...	1
F2 Coprocessor	Click to add executors...	1
F3 HouseKeeping - TMTC	Click to add executors...	1
F5 Memory Scrubbing	Click to add executors...	1
F6 OPSW Time Manager PPS Manager (25)	Click to add executors...	1
F7 FDIR EDR (26)	Click to add executors...	1
F8 OPSW Commandability TC Server (14)	Click to add executors...	1
F9 OPSW SMU If Non Pus Manager (15)	Click to add executors...	1
F10 Internal Interface (27,28)	ExecutionPlatform » Processors » LEON, ExecutionPlatform » Processors » MicroBlaze	1
F10T1 Parameter HK Manager (27)	ExecutionPlatform » Processors » LEON, ExecutionPlatform » Processors » MicroBlaze	
F10T2 Parameter Mom manager (28)	ExecutionPlatform » Processors » LEON, ExecutionPlatform » Processors » MicroBlaze	

Figure 21: Deployment model - Execution constraints

Then, every piece of information of this kind is labelled as an "execution constraint", and all these constraints are listed and attached to their corresponding tasks and flows.



- Deployment Constraints
  - Processor allocation
    - RTA - CPU Minimization
    - RTA With Offsets - CPUs Minimization
  - Feasibility analysis
    - RTA Manual Allocation With Offse
  - Task Partitioner
    - Task Partitioner - One To One
    - Task Partitioner - Many Tasks To O Thread
    - Task Partitioner - No Wait
  - Viewers
    - Deployment view

Task Partitions

Subsystem	Flow	Thread @ Allowed Executors	Task	Order	Priority
Reconf SW Module					
	F1 - Partial Reconfiguration Manager				
		Thread(63-1) @ LEON, MicroBlaze			
			T1 - Partial Reconfiguration Manager	0	0
LLSW Module					
	F4 WatchDog				
		Thread(64-1) @ LEON, MicroBlaze			
			F4T1 WatchDog	0	0

Figure 22: Deployment model - Task partition

Once a partitioning algorithm has been executed, it will be obtained a complete allocation of flows and tasks to available hardware resources as it can be seen in the previous image. In this example, a "One To One" allocation algorithm has been selected. At the right panel a final scheme of the task partition is shown after the allocation performed by the algorithm which has taken into account the predefined execution constraints to compute a full allocation of all the tasks and flows to the available hardware.

3.2 Detailed description of the application and the use case

Both execution platform and the application software have been modeled in their corresponding stages, manually mapping some flows to specific processors at the "deployment model" stage. Different versions of the Response Time Analysis algorithm are provided in order to balance load among processors, or to minimize the number of required processors.

Thanks to the sensitivity analysis, if a design error is found with a given analysis, the tool suite suggests a change in the application model parameters in order to achieve a feasible task set for the current execution platform and application model co-design.

Next a brief summary of the actions performed at each stage to complete a full cycle of modeling, design, analysis and code generation is presented to give simple idea of the steps required to design and validate an embedded real time system:

3.2.1 Platform Model description

A reduced set of hardware devices and its corresponding parameters can be modeled for each execution platform at the so called Platform Model stage. A compact graphical representation of the modeled platform is shown to the user as an intuitive linked graph. This functionality is shown at the "Board Model" panel and is aimed to easily found any hardware component incompletely defined, with no link to other components.

3.2.2 Application Model description

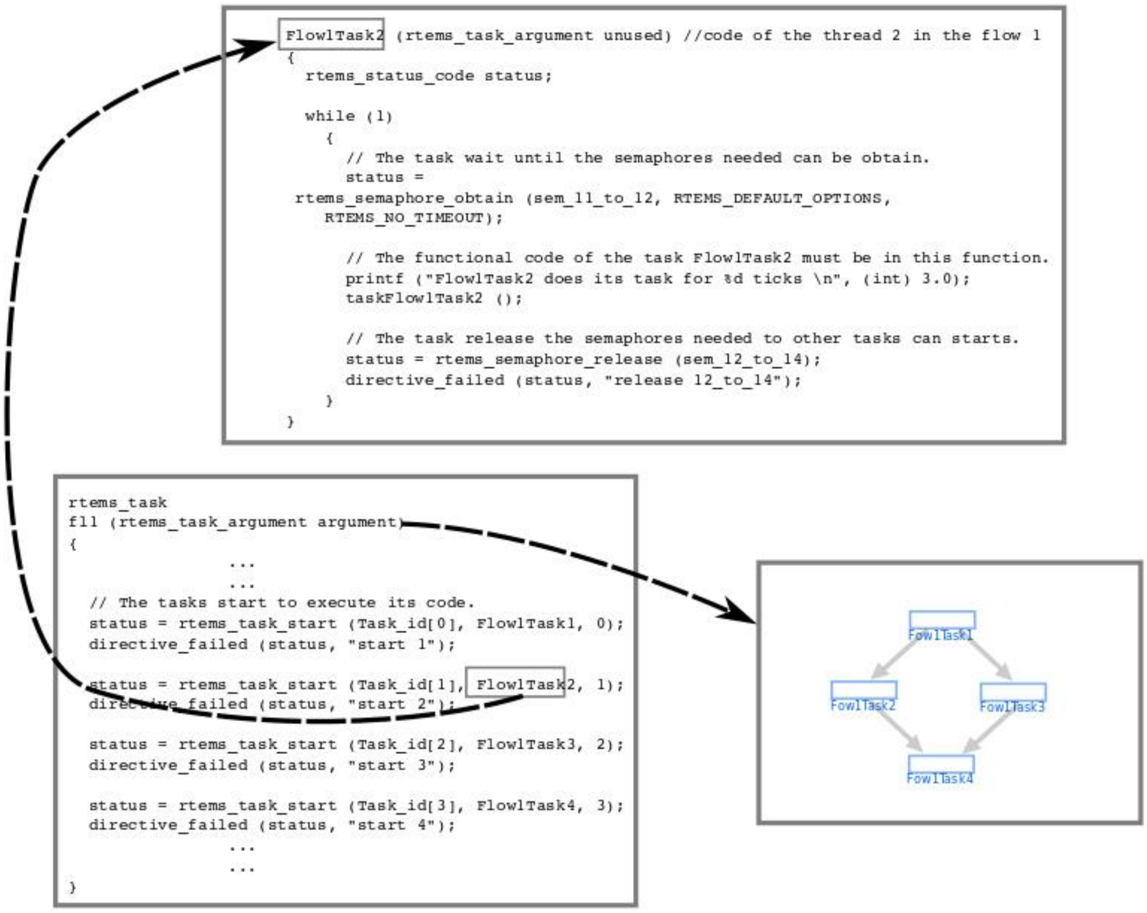
Systems, Subsystems, Flows and Tasks can be hierarchically defined in order to model a task set at the Application Model stage. As an example, a snapshot of a flow defined to model the UART utilization into the "Satellite" Use Case it is shown, where several flow parameters have been defined as <Period>, <Deadline> and <Priority>. Besides, a manual mapping of the <Processor> has been assigned.

3.2.3 Deployment Model description

Input data and the results provided by the tool suite are now better organized. Task and flow allocation to available hardware can be now perfectly separated from the platform and application models. Several task partitioning algorithms can be run to establish which software should be executed into each available hardware resource.

3.2.4 System Analysis description

The engineer is able to specify the kind of system analysis she wants to perform. It is also possible to interact with the tool to fix any issues that makes the system unfeasible. The implementation of these analyses is fast enough to be executed "on the fly", so the user can easily test several possible conditions and see the result immediately.



```
Flow1Task2 (rtems_task_argument unused) //code of the thread 2 in the flow 1
{
  rtems_status_code status;

  while (1)
    {
      // The task wait until the semaphores needed can be obtain.
      status =
 rtems_semaphore_obtain (sem_11_to_12, RTEMS_DEFAULT_OPTIONS,
    RTEMS_NO_TIMEOUT);

      // The functional code of the task Flow1Task2 must be in this function.
      printf ("Flow1Task2 does its task for %d ticks \n", (int) 3.0);
      taskFlow1Task2 ();

      // The task release the semaphores needed to other tasks can starts.
      status = rtems_semaphore_release (sem_12_to_14);
      directive_failed (status, "release 12_to_14");
    }
}
```

```
rtems_task
f11 (rtems_task_argument argument)
{
                ...
                ...
  // The tasks start to execute its code.
  status = rtems_task_start (Task_id[0], Flow1Task1, 0);
  directive_failed (status, "start 1");

  status = rtems_task_start (Task_id[1], Flow1Task2, 1);
  directive_failed (status, "start 2");

  status = rtems_task_start (Task_id[2], Flow1Task3, 2);
  directive_failed (status, "start 3");

  status = rtems_task_start (Task_id[3], Flow1Task4, 3);
  directive_failed (status, "start 4");
                ...
                ...
}
```

Figure 23: Code generation - RTEMS example

3.2.5 Code Generation description

Use Case 9.2 demonstrator had previously used a manually written example of RTEMS code in order to help with the automatic code generation development. Currently a new functionality to generate an automatic version of the RTEMS source code is avialable.

In the previous figure, source code has been automatically generated for a RTEMS target platform. It is just a simple example that fulfills the purpose of demonstrationg the potential of this feature in order to help the engineer in the process of HW and SW codesign. It is a feuture currently available from the web user interface of the art2kitekt tool suite.

4. Optical Payload Applications

4.1 Integrasys Satellite Communications Link Emulator

In this section it is found a description of Integrasys‘ Satellite Communications Link Emulator Demonstrator structure, results and the derived conclusions.

4.1.1 Structure

Integrasys‘ Satellite Communications Link Emulator for multicore platforms can monitor the quality of the transmitted carriers, check the status of the channel and track the potential interferences which degrade the performance of the service. We have incorporated parallelisation techniques for multicore platforms which improve the performance of the signal processing application running in the emulator.

Below one can find a block diagram of the whole emulator.

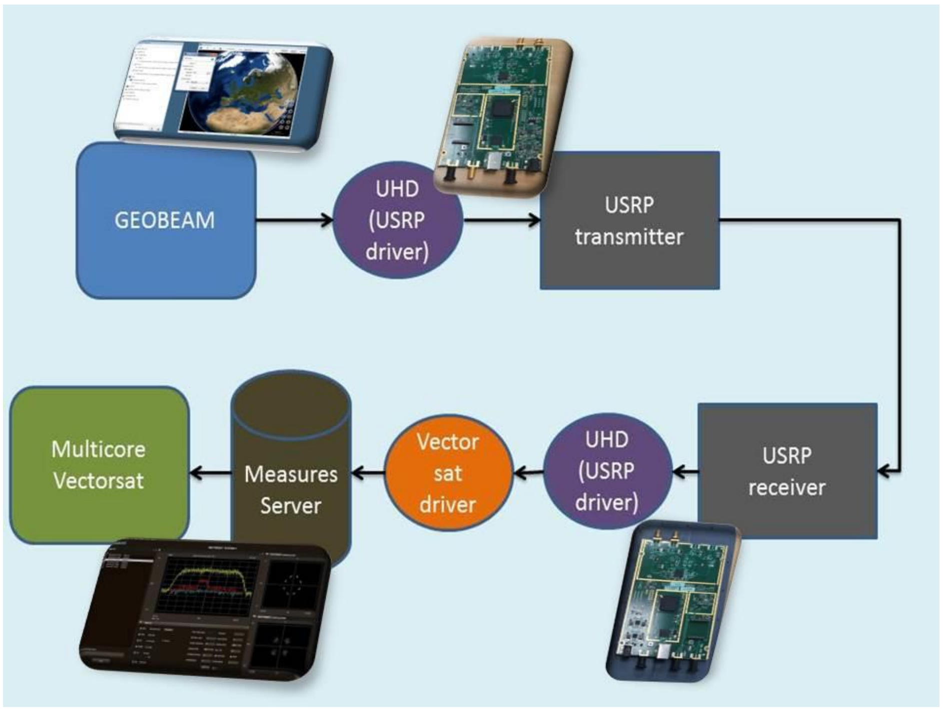


Figure 24: Satellite Communications Link Emulator Block Diagram

The emulator comprises the three main elements of a communications link:

- Transmitter
- Channel
- Receiver.

The Geobeam tool allows modelling a satellite communications network, with all the elements that are part of it, to achieve network planning and performance results. In the Emulator, GEOBEAM configures the link parameters and sends those parameters via a Socket connection to the UHD (USRP Hardware Driver), which communicates with the USRPs.

The hardware used to trasmit and receive the electromagnetic signals to simulate the carriers is the USRP B200 Software Defined Radio from Ettus Research.

The Vectorsat Server receives and processes the signal measurements from a RF device driver (in our use case, the USRP) and process all of them before sending them for visualization to the Vectorsat Client. Therefore, this module let us demodulate, visualise and analyse the signal that we have received on the second USRP, after having been modelled and sent with the first USRP.

4.1.2 Operation

We have performed several tests with the Satellite Communications Link Emulator in single core and multicore platforms to compare the improvement achieved when using more than one core. In this section we will summarize the test setup and the results achieved.

4.1.2.1 Test setup

For the tests we have used the following equipment:

- 1 Transmit USRP B200
- 1 Receive USRP B200
- 1 laptop for the GEOBEAM configuration module and connection to the USRP transmitter
- 1 single core platform for connection to the USRP receiver and evaluation of the receiving algorithms on a single core machine
- 1 Intel Core i3 dual core platform for connection to the USRP receiver and evaluation of the receiving algorithms on a dual-core machine.
- 1 Intel Core i7 quad core platform for connection to the USRP receiver and evaluation of the receiving algorithms on a quad-core machine.

Below, you can see a picture of a typical setup for one of the tests we performed.

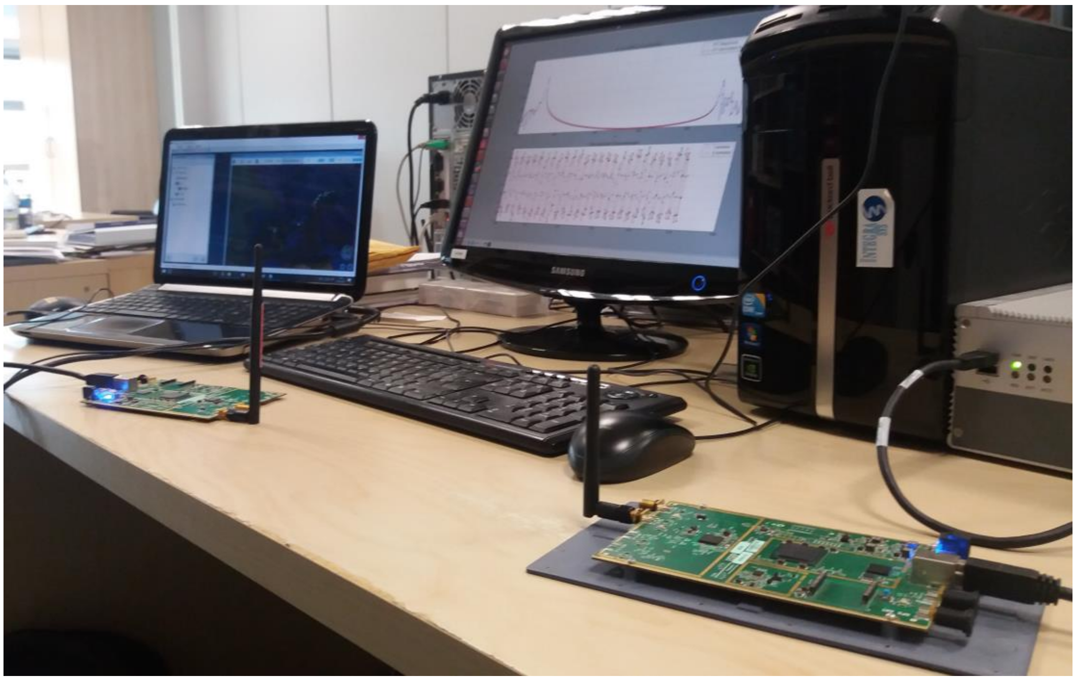


Figure 25: Application Test Setup

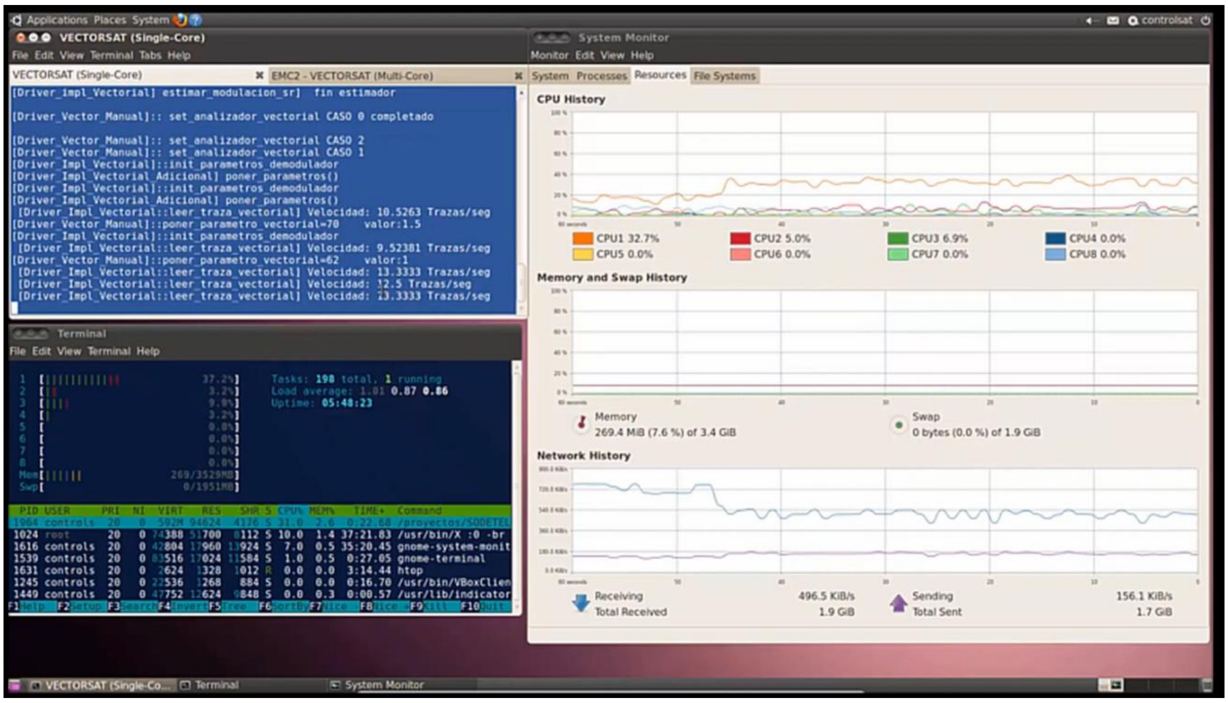
4.1.2.2 Test results

We transmitted typical RF signals with the first USRP and in tests we performed, the Vectorsat receiving application running on single-core mode, was able to able to provide 10 to 12 traces per second on 512 points for the FFT (Fast Fourier Transform) of the signal received on the second USRP. Then, when increasing the number of FFT points to 1024, performance decreased to 8 to 9 traces per second. These figures are somewhat poor in cases where a space application would require high resolution or multiple carrier support

On the other hand, the multi-core version running on dual and quad-core platforms, provided more than 20 traces per second (ranging from 25 to 28 traces per second), which makes more advanced and reliable features like several carrier simultaneous monitoring possible. This is achieved with the support of OpenMP to implement code parallelisation of the most costly signal processing operations for each core. Not all the benefits come from parallelisation as in this process we also had to refactor the code to make it parallelisable and follow some strategies in this line that made the code cache friendly.

In the following screenshots it will be shown a comparison between the resources monitor console results when running the receive application on a single core and a multi core platform.

In the first screenshot, we can see how in single core mode the traces per second that are being processed are around 10 (see upper left panel in blue and white).

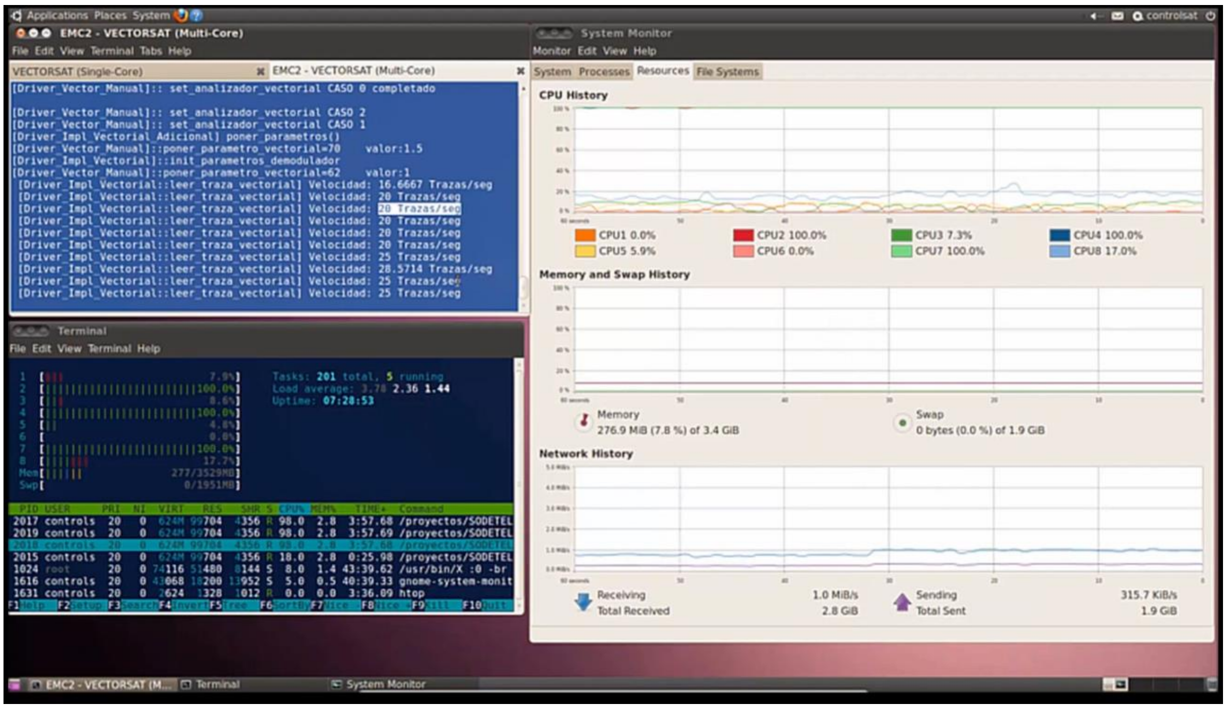


```
[Driver_impl_Vectorial] estimar_modulacion_sr]  fin estimador

[Driver_Vector_Manual]:: set_analizador_vectorial CASO 0 completado

[Driver_Vector_Manual]:: set_analizador_vectorial CASO 2
[Driver_Vector_Manual]:: set_analizador_vectorial CASO 1
[Driver_Impl_Vectorial]::init_parametros_demodulador
[Driver_Impl_Vectorial_Adicional] poner_parametros()
[Driver_Impl_Vectorial]::init_parametros_demodulador
[Driver_Impl_Vectorial_Adicional] poner_parametros()
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 10.5263 Trazas/seg
[Driver_Vector_Manual]::poner_parametro_vectorial=70    valor:1.5
[Driver_Impl_Vectorial]::init_parametros_demodulador
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 9.52381 Trazas/seg
[Driver_Vector_Manual]::poner_parametro_vectorial=62    valor:1
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 13.3333 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 12.5 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 13.3333 Trazas/seg
```

```
  1  [||||||||||||                 37.2%]     Tasks: 198 total, 1 running
  2  [||                            3.2%]     Load average: 1.01 0.87 0.86
  3  [||||                          9.9%]     Uptime: 05:48:23
  4  [|                             3.2%]
  5  [                              0.0%]
  6  [                              0.0%]
  7  [                              0.0%]
  8  [                              0.0%]
  Mem[||||||                 269/3529MB]
  Swp[                         0/1951MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1964 controls   20   0  592M 94624  4176 S 31.0  2.6  0:22.68 /proyectos/SODETEL
 1024 root       20   0 74388 51700  8112 S 10.0  1.4 37:21.83 /usr/bin/X :0 -br
 1616 controls   20   0 42804 17960 13924 S  7.0  0.5 35:20.45 gnome-system-monit
 1539 controls   20   0 83516 17024 11584 S  1.0  0.5  0:27.05 gnome-terminal
 1631 controls   20   0  2624  1328  1012 R  0.0  0.0  3:14.44 htop
 1245 controls   20   0 22536  1268   884 S  0.0  0.0  0:16.70 /usr/bin/VBoxClien
 1449 controls   20   0 47752 12624  9848 S  0.0  0.3  0:00.57 /usr/lib/indicator
F1Help  F2Setup F3SearchF4InvertF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 26: Resources Results on Single-Core Version

In the second screenshot we can see how in multi-core mode, in this case with four cores in use by the application, the figure is above 20 traces per second, therefore the improvement is clear.



```
[Driver_Vector_Manual]:: set_analizador_vectorial CASO 0 completado

[Driver_Vector_Manual]:: set_analizador_vectorial CASO 2
[Driver_Vector_Manual]:: set_analizador_vectorial CASO 1
[Driver_Impl_Vectorial_Adicional] poner_parametros()
[Driver_Vector_Manual]::poner_parametro_vectorial=70    valor:1.5
[Driver_Impl_Vectorial]::init_parametros_demodulador
[Driver_Vector_Manual]::poner_parametro_vectorial=62    valor:1
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 16.6667 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 20 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 20 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 20 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 20 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 20 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 25 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 28.5714 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 25 Trazas/seg
 [Driver_Impl_Vectorial::leer_traza_vectorial] Velocidad: 25 Trazas/seg
```

```
  1  [|||                       7.9%]     Tasks: 201 total, 5 running
  2  [||||||||||||||||||||||||100.0%]     Load average: 3.78 2.36 1.44
  3  [|||                       8.6%]     Uptime: 07:28:53
  4  [||||||||||||||||||||||||100.0%]
  5  [||                        4.8%]
  6  [                          0.0%]
  7  [||||||||||||||||||||||||100.0%]
  8  [|||||||                  17.7%]
  Mem[|||||||            277/3529MB]
  Swp[                     0/1951MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2017 controls   20   0  624M 99704  4356 R 98.0  2.8  3:57.68 /proyectos/SODETEL
 2019 controls   20   0  624M 99704  4356 R 98.0  2.8  3:57.69 /proyectos/SODETEL
 2018 controls   20   0  624M 99704  4356 R 98.0  2.8  3:57.68 /proyectos/SODETEL
 2015 controls   20   0  624M 99704  4356 R 18.0  2.8  0:25.98 /proyectos/SODETEL
 1024 root       20   0 74116 51480  8144 S  8.0  1.4 43:39.62 /usr/bin/X :0 -br
 1616 controls   20   0 43068 18200 13952 S  5.0  0.5 40:39.33 gnome-system-monit
 1631 controls   20   0  2624  1328  1012 R  0.0  0.0  3:36.09 htop
F1Help  F2Setup F3SearchF4InvertF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 27: Resources Results on Quad-Core Version

In the end the results are quite promising: compared with the initial single core version we got 130% of performance increase. The point is that the reduction of the computational costs for signal processing algorithms is even more important when we have more signal points to process.

4.1.3 Derived conclusions

Using multicore platforms for our signal processing application has left us with a clear improvement in performance and capabilities achieved. Obviously, the potential improvement is limited by the fraction of the software that can be parallelized to run on multiple cores but we have seen how implementing parallelization techniques for certain blocks of the signal processing code greatly increases the speed giving a much more efficient application which allows for much higher resolution and additional features, such as multiple carrier monitoring at the same time with less equipment that using the single core equivalent.

There are other derived benefits that can be proved from adopting multicore approaches in an application such as lower power consumption working at the same clock rate as in the single core variant and consequently less heat dissipation due to the lower energy used.

The disadvantages that we found were specially due to compatibility issues when trying to include third-party signal processing libraries used in the receiver application, especially when we tested more inexpensive and less widespread multi-core platforms.

4.2 MPSoC Video Processor for Earth Observation Instruments

Satellite image data compression is becoming important for new missions. Traditionally hardware implementation (FPGA) has been selected in order to provide high performance image processor solutions. As a result of the emerging multicores technologies it is possible to improve the software implementation performance by using scalable software implementation.

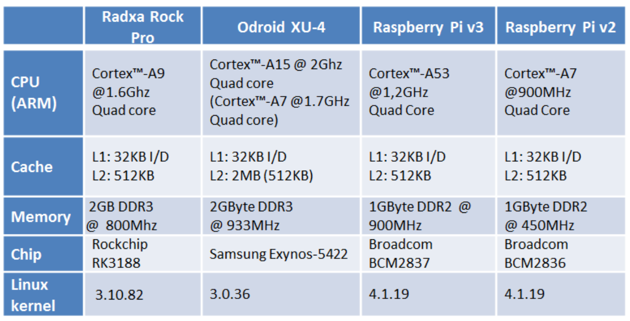
In order to take benefit of the new processors technologies a software parallel implementation of an image processor is presented. It consists on an image compressor based on CCSDS 122.0-B-1 standard for image data compression and an image cryptographic algorithm (AES-256-CTR) based on CCSDS 352.0-B-1 standard. Several parallelization paradigms were analyzed in terms of complexity, performance and portability. Finally, the OpenMP paradigm was selected and applied to Discrete Wavelet Transform, Bit Plane Encoding and AES cryptographic algorithm (counter mode).

The AES cryptographic algorithm has been implemented in software instead of using the dedicated hardware instructions available on the processors in order to port the implementation between several platforms.

4.2.1 Performance evaluation

In order to test the parallel version of CCSDS 122.0-B-1 and CCSDS352-B-1 a set of ARM multicore hardware platforms have been used. Three platforms use a quad-core processor (RK3188, BCM2837 & BCM2836) and the other one uses an octa-core (Exynos-5422). However the octa-core is based on Heterogeneous Multi-Processing (HMP) solution. Therefore we will only use the most powerful processors (quad-core Cortex-A15 @2GHz) in order to compare with the previous processors. Anyway, the differences between cores in Exynos-5422 (Cortex-A15 @2GHz vs. Cortex-A7 @1.7GHz) introduce a small improvement in terms of performance when we compare the speedup between four cores (Cortex-A15) and eight cores (Cortex-A15 & Cortex-A7).

The details of each hardware platform and linux kernel version are summerized in next table.



	Radxa Rock Pro	Odroid XU-4	Raspberry Pi v3	Raspberry Pi v2
CPU (ARM)	Cortex™-A9 @1.6Ghz Quad core	Cortex™-A15 @ 2Ghz Quad core (Cortex™-A7 @1.7GHz Quad core)	Cortex™-A53 @1,2GHz Quad Core	Cortex™-A7 @900MHz Quad Core
Cache	L1: 32KB I/D L2: 512KB	L1: 32KB I/D L2: 2MB (512KB)	L1: 32KB I/D L2: 512KB	L1: 32KB I/D L2: 512KB
Memory	2GB DDR3 @ 800Mhz	2GByte DDR3 @ 933MHz	1GByte DDR2 @ 900MHz	1GByte DDR2 @ 450MHz
Chip	Rockchip RK3188	Samsung Exynos-5422	Broadcom BCM2837	Broadcom BCM2836
Linux kernel	3.10.82	3.0.36	4.1.19	4.1.19

Hardware platforms based on ARM processors

The speedup obtained for each part of CCSDS 122.0-B-1 (DWT+BPE) can be seen in next figures.

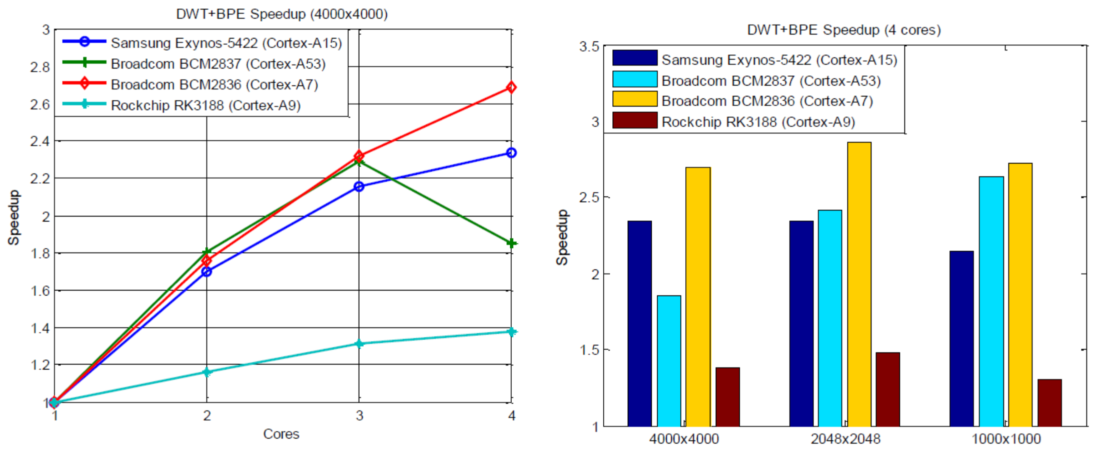


Figure 28: ***Speedup for DWT+BPE with a 4000x4000 image (8bit) & Speedup for DWT+BPE for three image sizes (8bits)***

There are minor differences when the speedup is computed with three image sizes (1000x1000, 2048x2048 and 4000x4000)

The throughput improvement is summarized in next table.

Cores	Rockhip RK3188 [Mpx/s]	Exynos 5422 [Mpx/s]	Broadcom BCM2837 [Mpx/s]	Broadcom BCM2836 [Mpx/s]
1	3.07	5.38	3.20	1.92
2	3.50	9.14	5.77	3.38
3	4.04	11.59	7.33	4.46
4	4.24	12.59	5.92	5.17

DWT+BPE: Throughput (Mpixel/s) with a 4000x4000 image (8bit)

In a similar way, the speedup of AES CTR mode has been computed with a 64Mbyte data chunk. The AES is a well suited algorithm for parallelization as can be seen in next figure where the scalability is close to be linear.

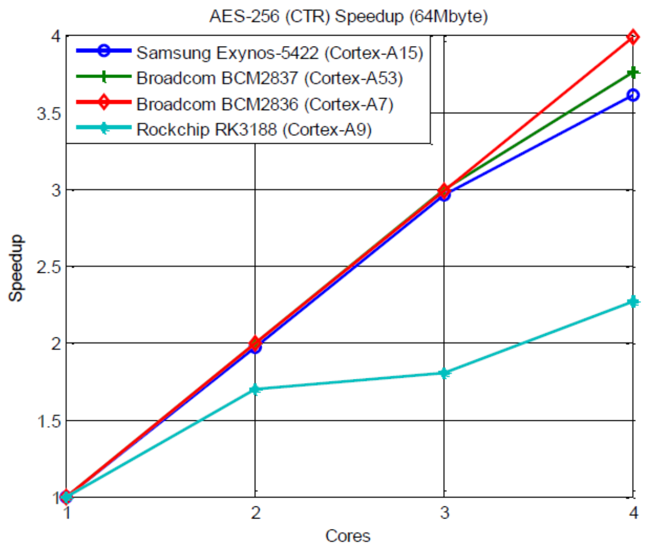


Figure 29: *Speedup for AES CTR mode*

The throughput of AES for each hardware platform is summarized in next table.

Cores	Rockhip RK3188 [Mbyte/s]	Exynos 5422 [Mbyte/s]	Broadcom BCM2837 [Mbyte/s]	Broadcom BCM2836 [Mbyte/s]
1	25.10	43.00	18.10	10.27
2	42.48	84.75	36.01	20.49
3	45.25	126.62	53.78	30.71
4	57.00	154.37	67.98	40.92

Figure 30: *AES: Throughput (Mbyte/s) with a 64Mbyte data chunk*

The scalability variation between processors for AES (linear) and DWT+BPE (nonlinear) reveals the dependency between the problem (algorithm) and the hardware architecture.

5. Use Case Platform Applications

5.1 Introduction

Multicore processors will be a key technology in the development of the next-generation of mini and micro satellites. In fact, these systems have stringent requirements from the point of view of size, weight and power consumption. The use of multicore platform allows to obtain a higher processing capability with lower footprint and energy requirements. Multiple software function can be allocated to the same processing element, avoiding the use of different boards based on a single-core dedicated processor.
In the context of task T9.4, TASI and Univaq investigated the possible advantages and drawbacks in using multicore processors for the development of satellite platform applications. Specifically, the possibility of using a single multicore platform to allocate different platform functions (telecommand and telemetry management, peripheral management, file transfer management etc.) has been considered. Since the different platform functions have different insurance level, proposed system shall be considered as a mixed criticality system.

The following critical issues have been considered:

- Resource partitioning: multicore processors are inherently characterized by an high degree of shared resources (system buses, memory bus and controller, shared cache memories, peripherals and logical units, etc.). This can lead to several problems in the case of hard real-time requirements, since the contention in access to shared resources may compromise the temporal determinism. Appropriate strategies in the management of access to resources and scheduling are therefore necessary.
- Isolation: a mixed-criticality system requires a strict isolation among functionalities with different criticality levels, both from the spatial (i.e. the access to the various resources) and temporal point of view (i. e. the execution times).

Virtualization appears as a feasible solution to guarantee both isolation and resource partitioning: in a virtualized environment, multiple application are able to run in isolated containers (virtual machines) and platform resources, as well as execution time, can be statically assigned to each single partition.

The task had the following specific goals:

- To evaluate multi-core platform for the aerospace domain, with specific reference to the Gaisler LEON4 based multicore platform.
- To evaluate the feasibility of virtualization-based software architectures and carefully benchmarking the performances of different hypervisors. SYSGO PikeOS e FentISS XtratuM have been considered as interesting solution for the aerospace domain.
- To implement a satellite prototypal platform based on the selected multi-core processor and hypervisors.

This document reports the final results of the analysis and details the development of the satellite platform prototype. Specifically:

- The quad-core Leon4 processor is analysed with specific reference to isolation support.
- The two hypervisor are analysed and benchmarked with respect to relevant metrics for the application scenario.

Proposed prototypal platform is described and implementation details are reviewed and analysed.

5.2 Target Hardware Platform

As detailed in Deliverable DL9.2, LEON4 processor has been selected as reference architecture for the implementation of the prototypal platform.
LEON4 is a 32 bit, SPARC-v8 compliant microprocessor representing the 4th generation of the LEON family, a widely adopted architecture in avionics and aerospace applications. The LEON4 processor,

promoted by the European Space Agency (ESA) and developed by Cobham Gaisler, is a candidate to be the next-generation microprocessor for satellites. The LEON4 is the basis of the radiation-hardened GR740 SoC, whose flight models are expected to be available within Q3 2018.
LEON natively supports multicore architectures, both in symmetric (SMP) and asymmetric multiprocessing (AMP). Moreover, the processor supports a wide range of peripheral devices providing additional processing and communication capabilities (for example, SpaceWire codecs and routers are directly supported by Gaisler).
A quad-core LEON4 configuration, as schematized in Figure 1, has been considered as a valid reference architecture for proposed application.

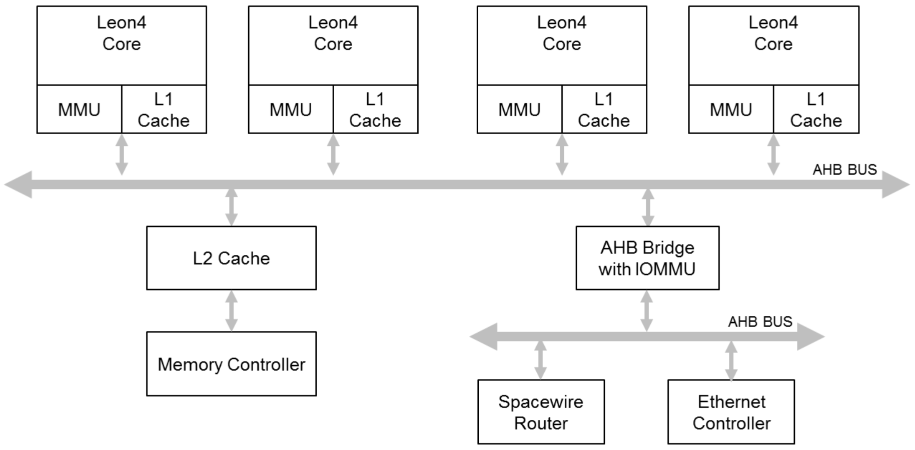


Figure 31: Simplified view of the Gaisler Leon4 processor

Two Cobham Gaisler development boards have been considered for this study, both mounting a quad-core LEON4 SoC, but implemented with different technology: GR-CPCI-LEON4-N2X [1] and GR-CPCI-XC4V on GR-RASTA avionics development platform. The following table details the relevant characteristics of the two boards.

Table 3: Overview of the target boards relevant characteristics

	GR-CPCI-LEON4-N2X	**GR-RASTA-LEON4**
Technology	eASIC Nextreme2	XC4VLX100-10FF1513C Xilinx Virtex 4 FPGA
Device ID / Build ID	0x280 / 4114	0x280, 0x281 / -
CPU	4 x LEON4 (No protection of L1 cache and register files)	4 x LEON4 (Protection of register files and L1 cache)
FPU	2 x GRFPU (shared per CPU pair)	4 x GRFPU (one per CPU)
CPU,FPU clock	150 MHz	45 MHz
Level-1 (L1) cache	I: 4x4KiB, D: 4x4KiB	I: 4x4KiB, D: 4x4KiB
Branch prediction	Yes	Yes
Memory Management Unit	Yes	Yes
Shared Level-2 cache	Yes	Yes

Level-2 cache size	4 x 64 KiB 256 KiB total	4 x 128 KiB 512 KiB total

The architecture of the SoC reflects the one schematized in Figure 1. It includes a shared AMBA AHB bus with two level of cache memories, a private L1 data & instruction cache and a shared L2 cache. The processors communicate with the peripheral devices by means of various bridges towards additional AMBA buses. High speed peripherals, like SpaceWire routers and Ethernet Controllers, are connected to an additional AHB bus by means of a AHB bridge.

As said, a relevant problem of multicore processing architectures is the presence of shared resources. The LEON4 data path includes three main potential source of interference: the AHB bus, the shared L2 cache and the central memory. Specific care shall be put in avoiding contention in accessing the shared resources.

LEON4 provides several mechanisms supporting isolation and predictability. For example: IOMMU: the IOMMU is able to map the device virtual DMA addresses to physical addresses. Moreover, it provides additional memory protection from peripheral ready and write operations. L2 cache partitioning: the LEON4 L2 cache supports a *master-index* mode in which one L2 cache way is assigned to each core, avoiding contention interference between cores.

5.3 Analysis and Benchmarking of the XtratuM and PikeOS Hypervisors

Different hardware and software solutions can be considered in order to guarantee space and time isolation between the different components of a mixed criticality software. Virtualization has been selected as a feasible technique to guarantee time and space isolation and resource partitioning. Two different hypervisors have been analysed on the target multicore hardware platform. The following sections details the analysis and benchmarking of the two selected software solutions:

- SYSGO PikeOS version 3.5 for SPARC v8 [3].
- FentISS XtratuM version 4.2.1 for LEON3/4 [4].

5.3.1 General architectural considerations

PikeOS is a real-time operating system able to provide paravirtualization services. As represented in figure 2, the PikeOS architecture is structured in four layers:

- Architecture & Platform support package: the lower layer of the PikeOS hypervisor, this two modules encapsulates respectively the details of the underlying CPU architecture and of the overall hardware platform. ASP and PSP offers a set of standard software interface for the execution of the PikeOS kernel on a specific platform.
- PikeOS kernel: a separation kernel supporting real time performance. PikeOS is based on a microkernel architecture directly inspired by the L4 kernel specification. The PikeOS kernel support tasks (virtual address spaces), threads (execution entities), thread scheduling and inter-thread communication. A Memory Management Unit is required in order to support a separate memory space for each task.
- PikeOS System Software, i.e. a module supporting various standard services (start-up of applications, file systems) and providing the partition abstraction. The PSSW is responsible for the creation of partitions and threads at boot time. The system configuration is stored in a dedicated database, the Virtual Machine Initialization Table (VMIT), specifying the partitions, their resources and access & control rights. PikeOS do not support dynamic process creation and the system configuration is statically specified in the VMIT.
- Partitions layer: a partition is an isolated set of applications executed under the predefined configuration specified in the VMIT. It should be noted that applications are able to use both PSSW API and system calls of the PikeOS kernel.

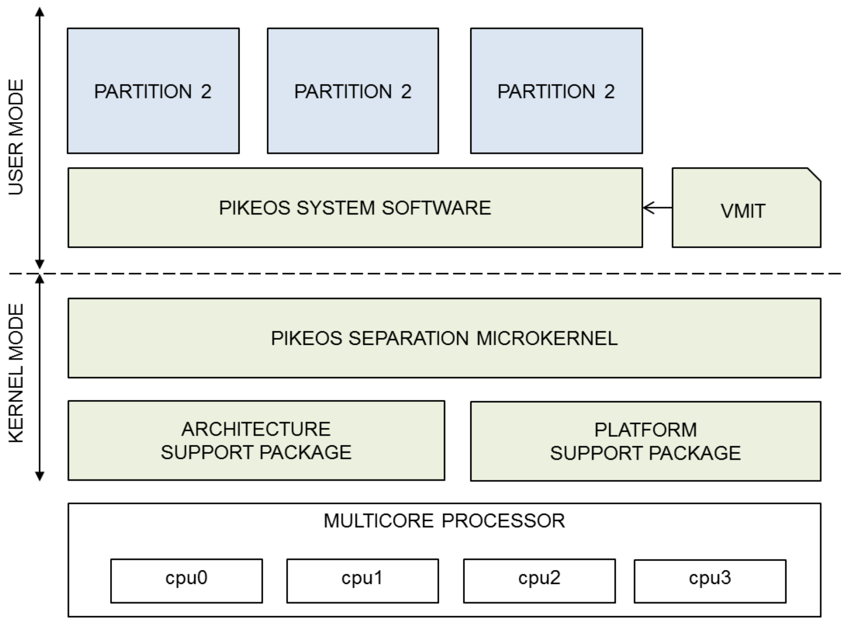


Figure 32: Schematization of the PikeOS hypervisor architecture

XtratuM is a bare metal hypervisor supporting paravirtualization. Originally developed as a Loadable Kernel Module for the Linux kernel, it evolved to a standalone software and has been ported to multiple architectures. XtratuM natively supports the SPARC architecture and LEON processors.

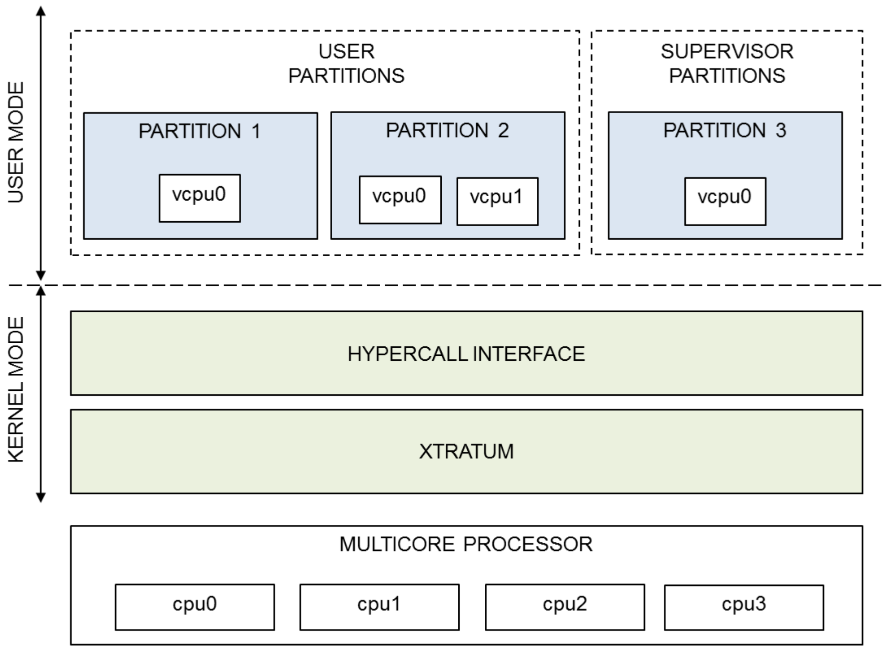


Figure 33: Schematization of the XtratuM hypervisor architecture

Figure 3 represents the architecture of the XtratuM hypervisor. The following elements are included:

- XtratuM kernel: a monolithic, non-preemptable kernel executed in the supervisor mode of the target processor. The kernel supports the virtualization of the hardware platform (i.e. the CPU, memory, interrupts, and critical peripherals), partition scheduling, inter-partition communication and health monitoring.

- XtratuM Hypercall Interface, i.e. the set of hypercalls used to access the paravirtualized services supported by the hypervisor.
- Partitions, i.e. the isolated execution environments. Like PikeOS, partition code has to be paravirtualized in order to run on top of the hypervisor. XtratuM do not support any native form of concurrency inside a partition.

5.3.2 Time management

In the standard configuration for the SPARC architecture, PikeOS timing is based on a system tick (*periodic ticker mode*) used for application timeouts, partition scheduling and system time-base. The PSP can be extended in order to support a tick-less mode (*dynamic ticker mode*), reducing the overhead of periodic tick handling and providing a finer time granularity. Moreover, dedicated hardware timers (if available for the platform) can be used for partition scheduling and system time-base, as well as for dedicated application level functionality (in this case, partition will require the access to the device and a dedicated device driver).
XtratuM adopts a different approach, virtualizing the hardware timer and providing two dedicated virtual timers to a partition:

- Virtual timer based on global clock, i.e. a timer based on the hardware clock.
- Virtual timer based on local clock, i.e. a timer based on a partition local time. This time only advances when the partition is actually running.

5.3.3 Performance considerations

Different metrics have been considered for the characterization of the hypervisors performance. In order to motivate some of the design choices detailed in the final section, the analysis of inter-partition communication mechanisms is detailed here.

In a virtualized environment, the different partitions are able to communicate, exchanging data and messages, only by means of the services provided by the hypervisor. Both PikeOS and XtratuM supports two different mechanisms for inter-partition communication:

- Messaging via sampling and queuing ports: this mechanism, directly inspired by the ARINC 653 specification, allows a message exchange between two different partitions by means of a specific communication primitive (a *channel* between the two partitions) statically defined at design time. Queuing ports define a FIFO style interface, whereas a sampling port only supports the storage of a single message (and an optional timeout for the validity of the message).
- Shared memory, i.e. a memory area accessible by two (or more partitions). The base address and size of the memory area shall be defined at design time. PikeOS also allow to define specific access rights for the memory area.

In the case of port based communication, the hypervisor is responsible for the encapsulation and transport of the message from the sender to one (or more) receiver partition.
In order to measure the overhead of sending and receiving messages using sampling and queuing port, a simple benchmark application has been defined. As illustrated in the following figure, two partition have been defined, a writer partition and a reader partition. The first partition is programmed to send a test message of 20 bytes length, and then become inactive. The second partition is programmed to receive the message.

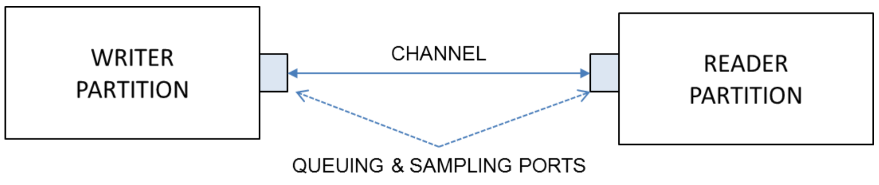


Figure 34: Inter-partition communication test setup

The following table reports the test results in the case of communication based on queuing ports and L2 cache disabled.

Table 4: Queuing port performance comparison, L2 cache disabled

	Send time [us]	Receive time [us]
Pikeos	1711	1712
XtratuM	191	202

The following table reports the results of the same test with L2 cache enabled.

Table 5: Queuing port performance comparison, L2 cache enabled

	Send time [us]	Receive time [us]
PikeOS	622	620
XtratuM	97	96

The following table reports the test results in the case of communication based on sampling ports and L2 cache disabled.

Table 6: Sampling port performance comparison, L2 cache disabled

	Send time [us]	Receive time [us]
PikeOS	1858	1864
XtratuM	149	199

The following table reports the results of the same test with L2 cache enabled.

Table 7: Sampling port performance comparison, L2 cache disabled

	Send time [us]	Receive time [us]
PikeOS	672	675
XtratuM	73	99

A similar setup has been evaluated in the case of communication based on shared memory (figure 5).

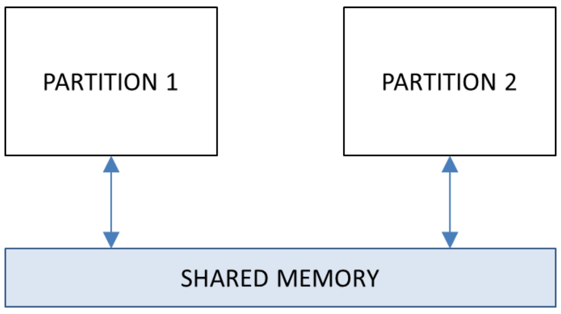


Figure 35: Shared memory performance test setup

The following table reports the test results.

Table 8: Shared memory performance comparison

	Write time [us]	**Read time [us]**
PikeOS	26	26
XtratuM	18	17

5.3.4 Consideration on the development of device drivers

PikeOS supports different strategies for the implementation of device drivers:

- Kernel mode device driver: this type of driver is implemented as part of the PSP and executed in supervisor mode. The driver is able to directly access the hardware resources, use dedicated kernel services and perform locking on a multicore platform.
- User mode device drivers: implemented as user level applications or PSSW extension, this type of drivers are executed in user mode and are thus preemptible. In this case, memory mapped device are directly mapped to the application virtual address space, whereas interrupts are managed using systems interrupt handling APIs.

XtratuM only supports partition level device drivers, and access grant to the peripheral devices shall be statically granted to the partitions. This partition will have direct control over the peripheral memory address space and is in charge of properly handling the device.
Kernel mode device drivers offers a better performance with respect user mode device drivers, but requires the driver to run in supervisor mode. Thus, the driver should guarantee the same level of insurance of the hypervisor itself (in real world scenarios, it should be certified at the same level of the hypervisor).
Nevertheless, in the case of PikeOS hypervisor the use of kernel mode device drivers has been considered for the final implementation due to notable performance limitations of the user mode mechanism natively provided by the hypervisor. As a reference, the following table details the time for reading and writing a 20-byte message in the case of an UART device.

Table 9: UART driver performance in PikeOS, kernel mode vs. user mode driver

	Write time [us]	Read time [us]
User Mode Driver	1603	688
Kernel Mode Driver	60	65

5.3.5 Re-use of legacy code

As said, the possibility of re-using legacy code is an important feature of virtualization. Both XtratuM and PikeOS allows legacy application to run inside isolated partitions, facilitating the porting to the new hardware platforms.
The possibility of code re-use has been verified by integrating into a dedicated partition a legacy software for the management of a Star Tracker device. The software was originally developed for a single-core, LEON3 based system and is capable of managing the various sensor functionality and process the measured data.
The Star Tracker interfaces over a SpaceWire link, and the control software shall be able to communicate over this interface. The following figure represents the integration of the software inside a virtualized partition environment:

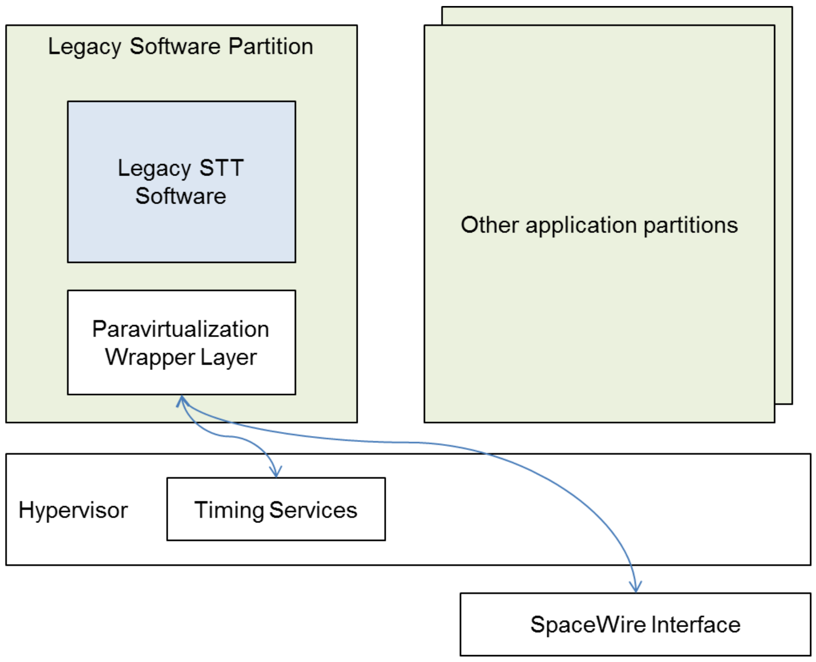

Legacy Software Partition
Legacy STT Software
Paravirtualization Wrapper Layer
Other application partitions
Hypervisor
Timing Services
SpaceWire Interface

Figure 36: Integration of a legacy Star Tracker management software in the virtualized environment

As illustrated, the integration of the STT management software required the development of a dedicated wrapper providing the following paravirtualized services:

- Low level access to the SpaceWire interface: as detailed in the previous section, access to physical devices requires the use of dedicated hypervisor services and hypercalls. A legacy software shall then be adapted in order to exploit these services, by modifying or wrapping every access to the peripheral memory space.
- Timing functionalities: the integration of the STT software required the adaptation of timing related functionalities, whose implementation has been based on timing services provided by the hypervisor.

5.3.6 Isolation related features

Both hypervisors supports time and space partitioning. Specifically, they allow to define:

- A set of partitions, i.e. isolated container or execution environments for the various applications. Each partition has access to a specific set of resources, statically defined at configuration time.
- A periodic cycle used to schedule the various partitions over time. A partition is able to run only in specific time intervals, statically defined at configuration time by means of the periodic cycle specification.

With specific reference to the multicore scenario, PikeOS maintains a single major time frame, shared by all the CPUs and periodically repeated throughout the operations. PikeOS distinguishes the concepts of resource partitions and time partitions, and different resource partitions can belong to the same time partition (in this case, resource partitions are scheduled by priority).

XtratuM allows the definition of a different cyclic plan for each CPU of the multicore platform. The basic scheduling unit of a multicore scenario is the so-called *virtual CPU*, and two scheduling policies are supported: cyclic scheduling, in which partitions are scheduled in a fixed, cyclic fashion, and priority scheduling, in which partitions are scheduled on the basis of partitions priority.

With respect to the detailed platform isolation-related features:

- LEON4 IOMMU is natively supported by the XtratuM hypervisor, whereas PikeOS SPARC PSP do not include a native support for IOMMU.

L2 cache management and configuration is not supported by the two hypervisors.

5.4 Prototypal Platform

A prototypal demonstrator representative of a simplified satellite platform has been implemented in order to evaluate the performances of the proposed approach. Specifically, designed system aims to model:

- the satellite's telecommand and telemetry function.
- the management of peripheral devices.
- the management of large file transfers.
- the execution of legacy STT code.

Proposed functionalities are simplified version of actual satellite functions: from PUS standard based Telecommand & Telemetry management, Satellite Housekeeping, Attitude & Orbit Control, Mass Memory Management etc. The prototypal platform has been described in details in deliverable DL9.4, and is schematized in the following figure.

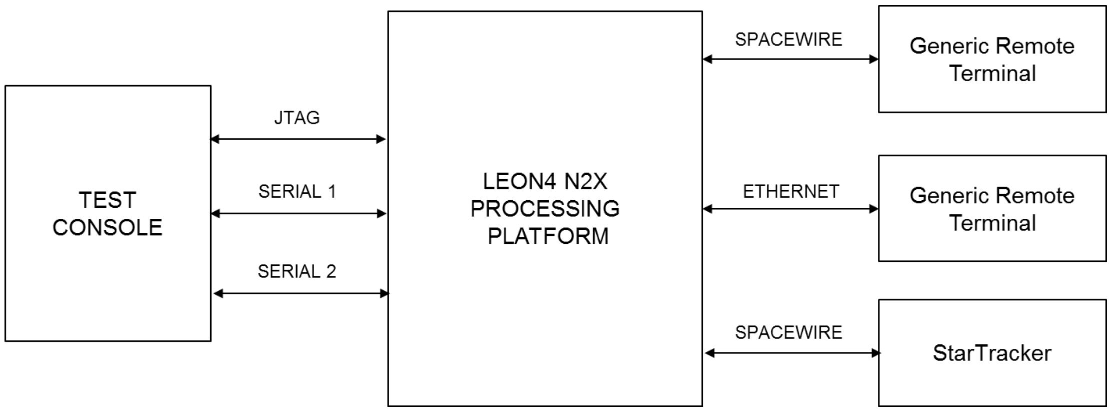


Figure 37: Proposed demonstrator setup

As said, the LEON4 multicore processing platform is based on a COTS board (Cobham-Gaisler GR-CPCI-XC4V or GR-CPCI-LEON4-N2X) including a quad-core LEON4 processor. The board acts as a satellite Platform Control Computer and execute the reference Command & Data Handling application software.
The following communication interfaces are used:

- SpaceWire bus as the main communication interface. Selected reference platform support SpaceWire communication by means of an 8-port SpaceWire router.
- Ethernet interface as secondary communication bus.
- Serial interfaces, used as a standard interface for communication and debug

A Test Console manages the Leon4 platform, sending telecommands, receiving telemetries and uploading and downloading data according to predefined traffic profiles. The serial port acts as TCTM interface for transmission and reception of telecommand and telemetries. Remote terminals and peripherals communicate over the SpaceWire Network, and by means of the Ethernet interface.
The system is able to manage different types of peripheral devices. Proposed setup includes both generic Remote Terminal emulators (useful to verify test communication links and verify predefined communication patterns) and actual satellite components. A Star Tracker device is used to analyse the performance of the platform with respect to real world applications and to show the hypervisor capabilities in reusing legacy code.
Figure 9 show a logic schematization of prototype software. The following components are included:

- I/O Management: a component in charge of managing access to peripheral devices and guaranteeing the correct timing of the various transactions.
- TC & TM Management: a component in charge of managing telecommands and telemetries. This component will receive from the I/O manager the telecommands sent to the platform, check their correctness and forward them to their destination components.
- STT Management: a component in charge of managing a Star Tracker device, receiving and processing its data and generating the correspondent command sequences. This component is based

on a legacy management software, as described in section 3.5. The TCTM manger is able to forward to this component the commands received from the test console.

- Large File Transfer Management: a component in charge of managing large file transfers between different I/O interfaces (accessed by means of the I/O manager). The TCTM manger is able to forward to this component the commands received from the test console.

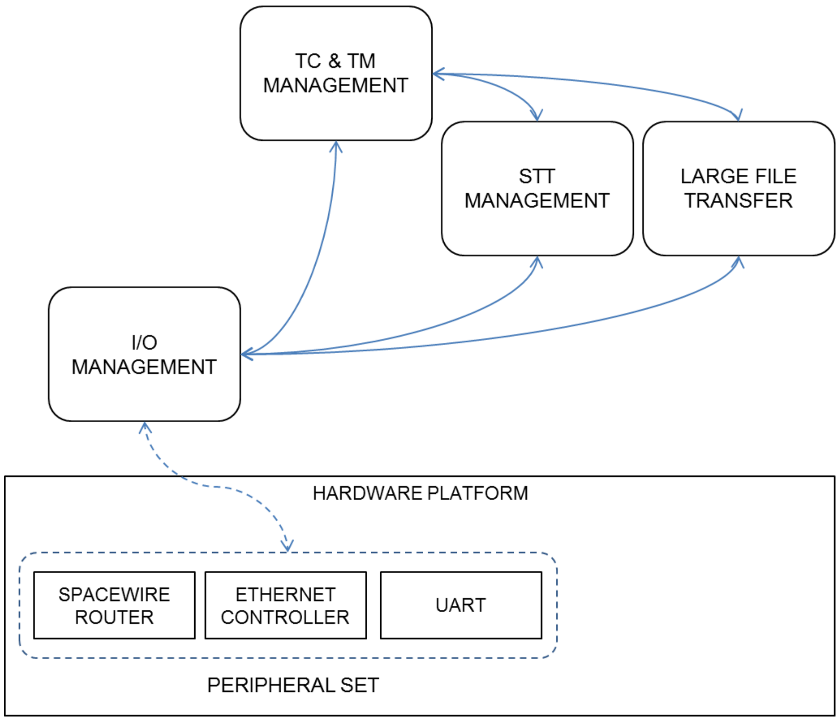


Figure 38: Prototype software logic schematization

Detailed components are characterized by different criticality levels, as resumed in the following table (higher level correspond to higher criticality):

Table 10: Criticality levels for the various software components

Criticality Level	Components
3	I/O Management
2	STT Management, TC&TM management
1	Large File Transfer Management

5.4.1 Design details

As said, all software components rely on a I/O management component to access the external devices. This component includes two different sub-components:

- A *Transaction Router*, handling the RMAP requests coming from the different on-board software entities, and forwarding them to the Spacewire interfaces (by means of the I/O Scheduler).
- An *I/O Scheduler*, forwarding the data provided by the device interfaces and taking care of timing requirements.

The I/O Scheduler, implemented as a table-driven cyclic executive, is able to manage the access to the various interfaces in a strictly deterministic way. This component supports three different transaction typologies: high priority transactions with hard real time requirements, medium priority transactions with soft real time requirements and low priority transactions, with no specific real time requirements. The management of different transaction priorities has been already detailed in deliverable D 9.5.

The scheduler defines:

- A series of elementary frames (bus time slots), each one dedicated to the management of one or more transaction.
- A major cycle, constituted by the succession of the various bus time slots and cyclically repeated.

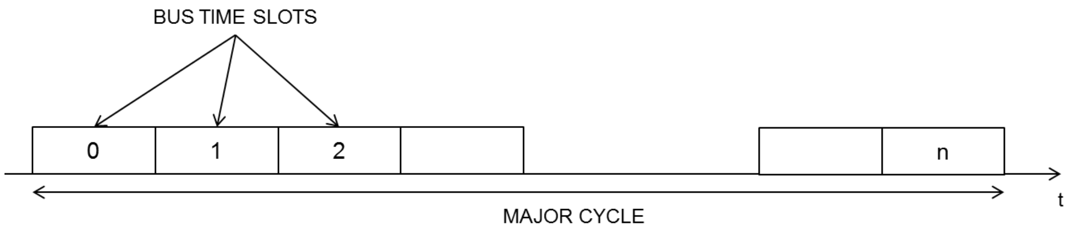


Figure 39: Major and minor cycle for the I/O Scheduling

As said, the scheduler is table driven: the operations to be executed in each Bus Time Slot are statically defined at design time.
The following figure specifies the implementation of the telecommand & telemetry management and its interaction with the I/O management, detailing the various sub-components involved.

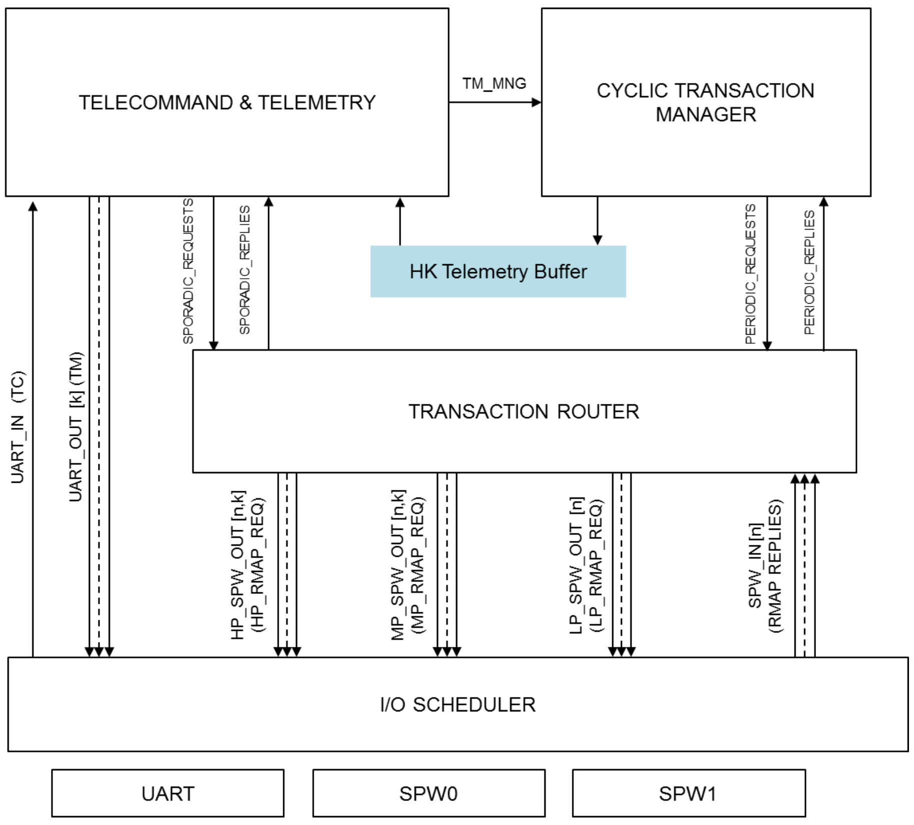


Figure 40: Logic schematization of the Telecommand & Telemetry management function

The Telecommand & Telemetry component include the following sub-components:

- TCTM Manager: in charge of managing all the telecommands received from the test console and of generating and sending the platform telemetries. This component also manages the activity of the Cyclic Transaction Manger.
- Cyclic Transaction Manager: in charge of periodically acquire values and generate telemetries from peripheral devices.

Both sub-components are able to send request to SpaceWire remote terminals using the RMAP communication protocol.
The different sub-components communicates by means of message passing channels, implemented as FIFO queues, and shared memories areas, statically defined at design time.
Specifically:

- The TCTM Manager is able to receive telecommands and send back telemetries communicating with the I/O Scheduler by means of dedicated communication channels.
- Each component managing RMAP request will have one sending channel and one receiving channel connected to the Transaction Router.
- The I/O Scheduler allows to manage the different priority levels and timing by exposing multiple communication interfaces to the Transaction Router. Specifically, the *(n,k)* channel will be associated to the transactions over *n-th* interface, scheduled to the *k-th* time slot.
- The Cyclic Transaction Manager will store the periodically acquired data inside a shared memory area (Telemetry Buffer), that will be read by the TCTM Manager.

The following figure specifies the implementation of the Star Tracker management function and its interaction with the I/O management, detailing the various sub-component involved.

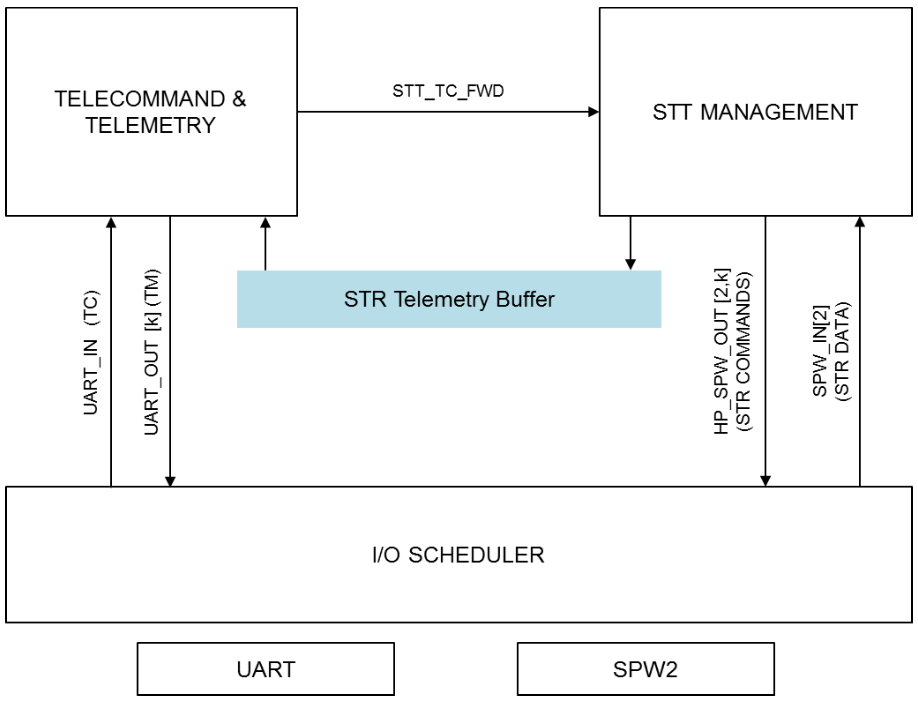


Figure 41: Logic schematization of the Star Tracker management function

As schematised, a single sub-component is in charge of STT management. The STT manager is able to periodically read the device data and generate the related command sequences. The user can manage the sub-component by sending telecommands, that are preliminarily checked and forwarded by the TCTM over a dedicated message buffer (STT_TC_FWD). The sub-component is also able to process the sensor data and generate related telemetries. A dedicated telemetry storage area is used to store generated telemetries, which are read and sent back to the console by the TCTM.
The following figure specifies the implementation of the Large File Transfer management function and its interaction with the I/O management, detailing the various sub-components involved.

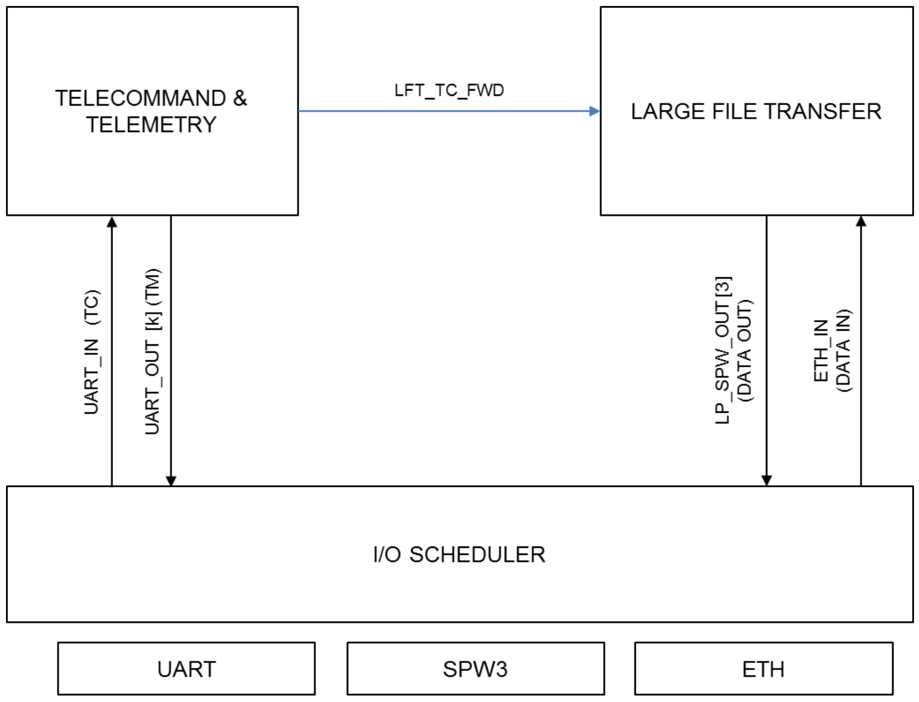


Figure 42: Logic schematization of the Large File Transfer management function

A dedicated sub-component is in charge of Large File Transfer management. This object is able to query data from one of the interface and forward them to another interface, and is allowed to continuously perform background (low priority) transactions. The sub-component is managed by means of dedicated telecommands, preliminarily checked and forwarded by the TCTM over a dedicated message buffer (LFT_TC_FWD).
Table 9 details a reference timing specification or the various sub-components:

Table 11: Reference timing for the proposed prototype platform

	Period [ms]
TCTM	125
Cyclic Transaction Manager	125
STT Manager	125
Large File Transfer	125
Transaction Router	125
I/O Scheduler	7.8125

5.4.2 Hypervisor-based implementation

A preliminary allocation of the various modules to different partition has been obtained by mapping the different modules to different partitions, each one accessing a specific set of dedicated resources, statically allocated by means of the hypervisor services.

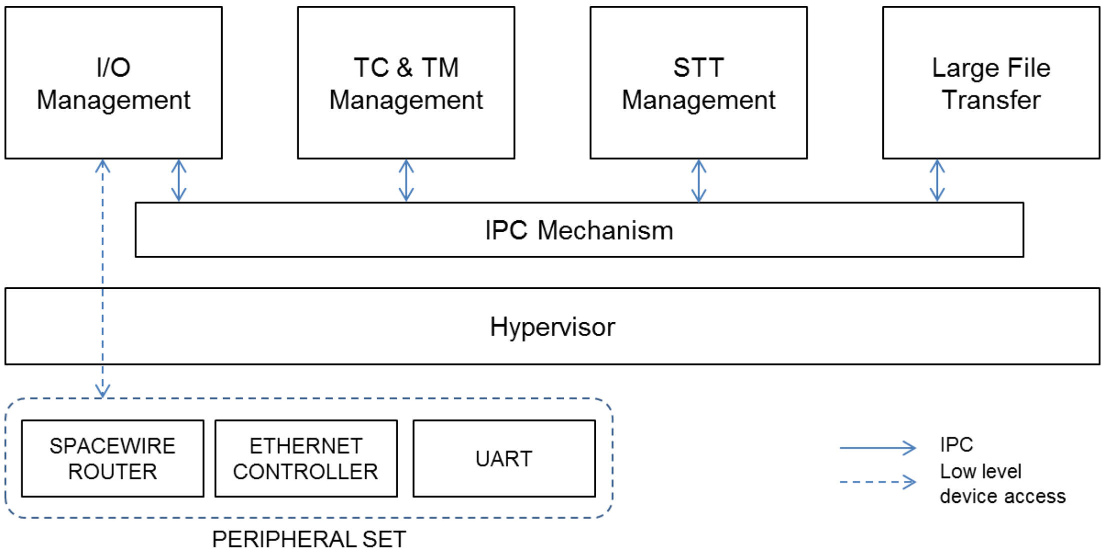


Figure 43: Hypervisor based implementation of the proposed software

As described in DL9.5, the various software sub-components have been implemented as periodic tasks, adopting a scheduling policy directly inspired by the ADA Ravenscar profile model. The following figure shows the generic task execution model.

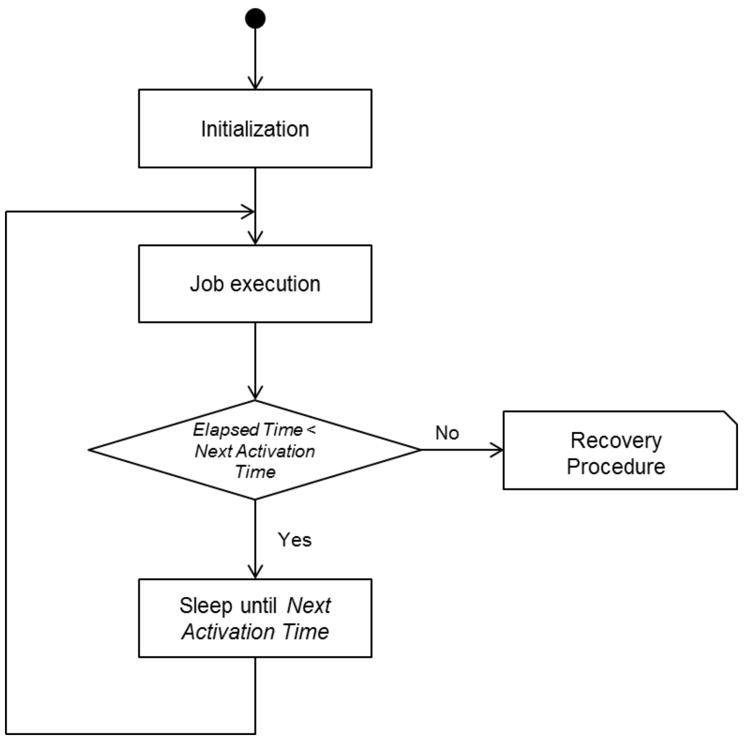

Initialization
Job execution
Elapsed Time < Next Activation Time
No
Recovery Procedure
Yes
Sleep until Next Activation Time

Figure 44: Generic task execution model

Each task is characterized by:

- A job, consisting in the activities to be performed.
- An initial activation time, statically defined by design. The task will start to execute its job for the first time at this specific instant.
- A period, i.e. the interval between two successive task activation.
- A priority, which determine the scheduling of the various tasks.

Each task will have an execution deadline, as it shall complete the job before its next activation time. If a task misses its deadline, a software-based recovery action is triggered. A task is created and started using the specific hypervisor's functionalities. It is assumed that each task is able to complete the initialization phase before its initial activation time.

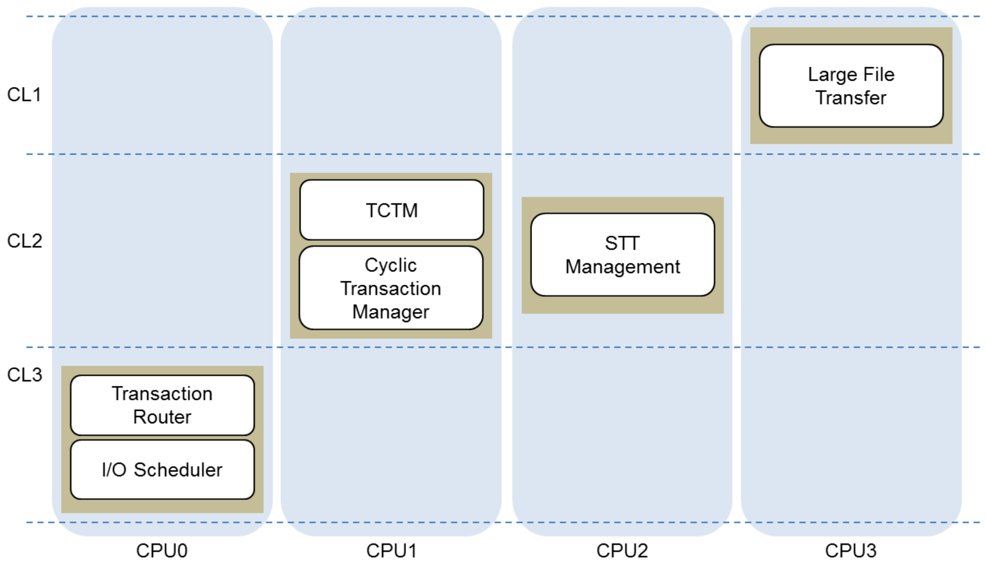


Figure 45: Mapping of the various software component on different CPUs with different partitions

Figure 15 schematized the detailed allocation of the various sub-components, per criticality and CPU. The following design choices have been made:

- Different components have been statically mapped to different CPUs.
- Different sub-components have been mapped to different criticality levels.

The following requirements have then been enforced:

- Only sub-components with the same criticality level shall compete for accessing the platform shared resources.
- Only the I/O scheduler is enabled to access the external devices.
- Component at the lowest criticality level are only enabled to execute background transactions.

5.4.3 PikeOS specific approach

The following figure schematizes the implementation of the reference software using the PikeOS hypervisor.

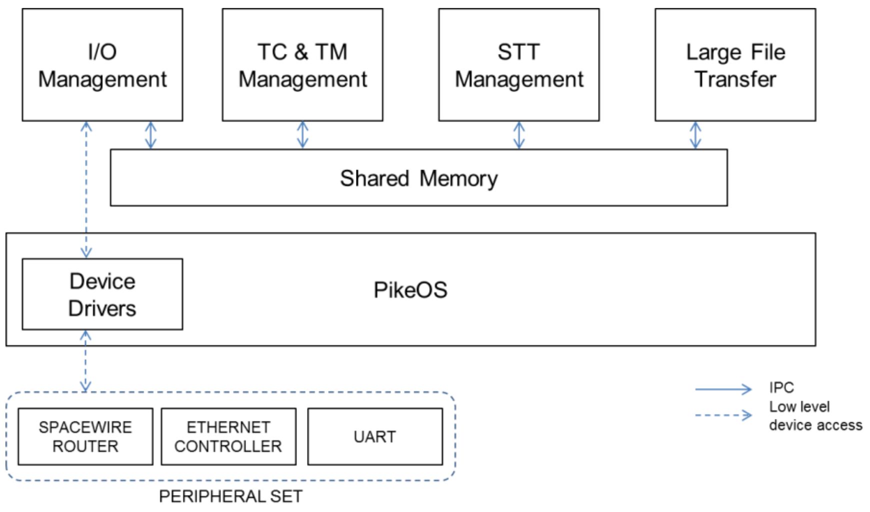


Figure 46: PikeOS based implementation of the proposed software

The following design choices have been made for the development of the PikeOS based version of the application:

- Mapping of different sub-components on single-task, single-thread PikeOS applications. The specific sub-component implementation reflects the behavior detailed in Figure 14.
- Use of custom inter-partition communication mechanisms. This has been motivated mainly by the application performance requirements.
- Implementation of kernel-level device drivers, in order to improve the system responsiveness.

5.4.4 XtratuM specific approach

The following figure schematizes the implementation of the reference software using the XtratuM hypervisor.

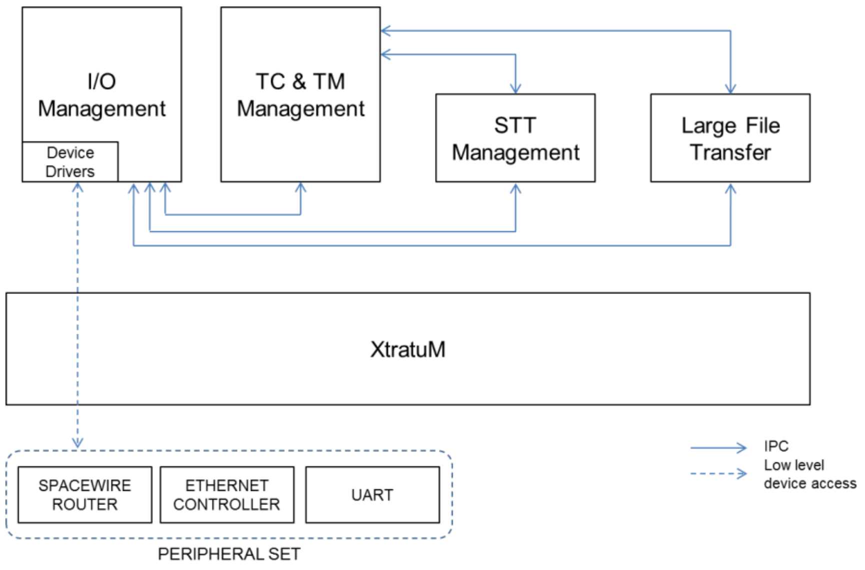


Figure 47: XtratuM based implementation of the proposed software

The following design choices have been made for the XtratuM-based development:

- Development of a custom partition-level tasking abstraction for multitask partitions.
- Use of native inter-partition communication mechanisms.
- Use of partition level device drivers.

5.5 Conclusions

TASI and Univaq activity for Task T9.4 focused on the analysis and benchmarking of a multicore processing platform based on Gaisler Leon4 processor, and of hypervisors and paravirtualization as a feasible solutions supporting the development of a prototypal avionic software. Two different hypervisors have been analysed and tested, SYSGO PikeOS and FentISS XtratuM.

The study allowed to compare the different software solutions and to analyse the performance of Leon4 processor. Specifically, the following results have been achieved:

- Development of a Platform Support Package for the PikeOS hypervisor on quad-core Leon4 processor;
- Development of UART and SpaceWire drivers for the PikeOS hypervisor, both in the user mode version and kernel mode version.
- Development of UART and SpaceWire drivers for the XtratuM hypervisor (user-mode version only, as XtratuM does not support the integration of kernel-mode drivers).
- Analysis and benchmarking of hypervisor performances, with specific focus on device driver performance and inter-partition communication.
- Analysis of legacy software reuse by means of the integration of a Star Tracker management library with the PikeOS hypervisor. The library was originally developed on the Leon3 processor.
- Development of a prototypal avionic software supporting Telecommand & Telemetry Management and Large File Transfer Management.
- Development of a demonstrator based on mentioned hardware and software solutions. Integration of the platform with peripherals and analysis tools.

6. Radar Payload Applications

6.1 Algorithm Requirements for the HPEC Architecture Study Case

Modern Monitoring-&-Forecasting Geoscience-Services as well as Governmental-Operations for the Homeland Protection (HP) demand an ever increasing performance of remote-sensing capabilities from space. Accordingly the evolution of Spaceborne Radars (SBRs) proceeds at small, yet steady, steps towards improvements in terms of Mission Aspects, Signal Processing Techniques, Payload Architectures, and related (possibly blue product line) Enabling Technologies. More specifically future SBR Payloads requirements are evolving in strict correlation to the trends in specific aerospace and electrical engineering fields sprouting towards flexible, modular, interoperable, and cost-effective payloads subsystems.

The key concept for this strategy relies on tactics aimed at exploiting advanced Analysis, Simulation, and Breadboarding activities for Knowledge Aided (KA) Designs and Distributed High-Performance-Extreme-Computing (HPEC) digital architectures based on large and fast Shared-Memories. In particular Multicore-Microprocessors should be aimed either at Job-Level Parallelism via multiple processors running independent activities i.e. Multiple-Instructions-Multiple-Data (MIMD) or at Parallel Processing via multiple processors running simultaneously the same activity i.e. Single-Instruction-Multiple-Data (SIMD).

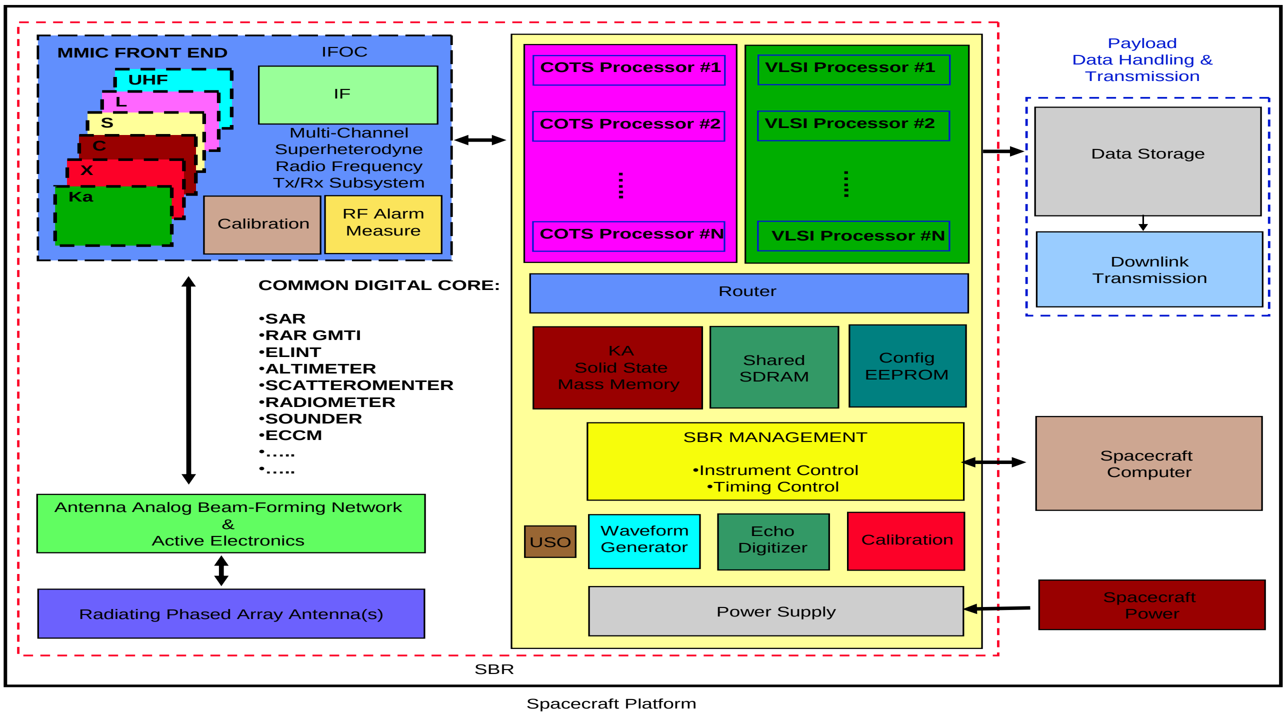


Figure 48: High Level Block Scheme of SBR Payloads

This application paves the way for such SBR challenges providing proof-of-concept of the aforementioned Distributed HPEC architecture based on a Large-Shared-Memory taking as a reference a preliminary and scaled study-case for Onboard Raw-Data Acquisition for novel Multi-Channel Synthetic Aperture Radar (SAR) High-Resolution-Wide-Swath (HRWS) modes comprising Elevation Digital Beamforming (DBF) processing, Azimuth Multi-Channel processing, and Block Adaptive Quantization (BAQ) processing on dummy digital inputs.

6.2 EMC2 Demonstrator Functional Requirements

A significant study-case has been selected to verify the design, implementation, and performance of an EMC2 demonstrator as a representative of the EMC2 digital architectural core,
The digital breadboard accounts for a SAR instrument operating in a High Resolution Wide Swath (HRWS) mode. The relevant requirements are defined in the following sub-sections.

This type of instrument and operative mode have been selected for the following reasons:

1. The required set of functionalities is characterized by a computational complexity resembling a worst-case scenario suitable to verify the EMC2 demonstrator performance.

2. The architecture along with the algorithms are innovative in the frame of Spaceborne Radar systems.

3. The processing includes the Digital Beam Forming (DBF) technique which is one of the key on-board functionalities for next-generation Spaceborne Radar systems along with Multibeam, Scan-On-Receive, Moving Target Identificator, Multi-polarimetry and Side-lobe Nulling.

During the development phase, the available hardware resources, which might be limited because of budget reasons, might not be sufficient to keep up with the requested computational effort. In this case, a reduced set of functionalities shall be selected for the implementation and proper parameter scaling shall be applied in order to cope with the actual HW capabilities. An assessment on the capabilities of the final EMC2 demonstrator configuration shall be expected.

6.2.1 **HRWS Introduction**

The HRWS SAR mode is based on a multi-static Spaceborne Radar concept that allows improving both the azimuth resolution and the swath width during a continuous ground coverage in stripmap mode. This is an evolution with respect to conventional monostatic SAR systems whereas the two features pose contradicting requirements: a fine azimuth resolution requires a high Pulse Repetition Frequency (PRF), while a large unambiguous swath requires a low PRF.

The HRWS technique allows overcoming this limitation by using two separated antennas for transmit and receive functions:
- A transmit antenna having a smaller length *l* and height *w* compared to the receive antenna.
- A receive antenna having a larger length L and height W which is split into multiple sub-apertures, N in the azimuth dimension and M in the elevation dimension, as schematized in next figure. Accordingly, the subapertures receive the backscattered signal simultaneously. Each sub-aperture comprises a different receive channel, whereby the received signal is frequency down-converted to baseband and digitized by an Analog-to-Digital Converter (ADC).

Such a design allows obtaining the desired swath width and fine azimuth resolution. The former is in fact unambiguously determined by the transmit antenna size for a given PRF while the latter can be obtained at the same PRF by proper multi-channel sampling of the azimuth spectrum. The Azimuth DBF Processing requirements are defined in the following section6.2.2.2

A DBF processing on receive is carried out in the elevation dimension in order to obtain multiple "pencil" beams, steered in elevation while the signal is backscattered from the earth surface. This technique, called Scan-On-Receive (SCORE), allows each resolution cell of the target area being illuminated with the peak gain of the receive antenna, thus compensating for the smaller transmit gain as shown in next figure.
The elevation DBF processing requirements are defined in the following section.

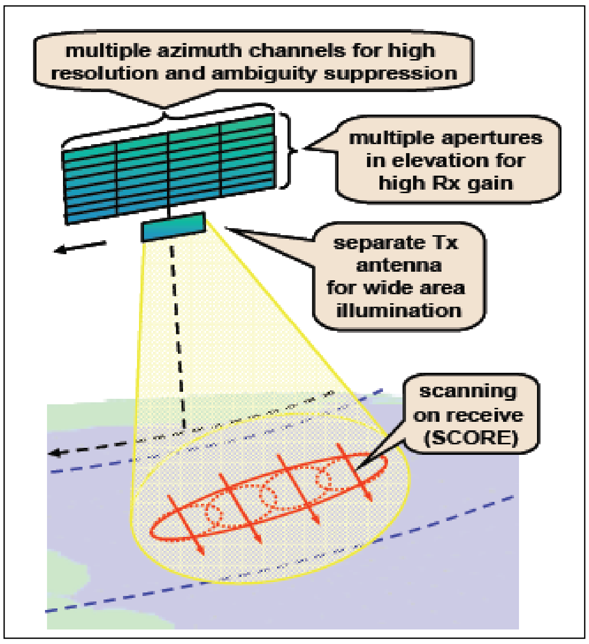

multiple azimuth channels for high resolution and ambiguity suppression
multiple apertures in elevation for high Rx gain
separate Tx antenna for wide area illumination
scanning on receive (SCORE)

Figure 49: High Resolution Wide Swath (HRWS) SAR System.

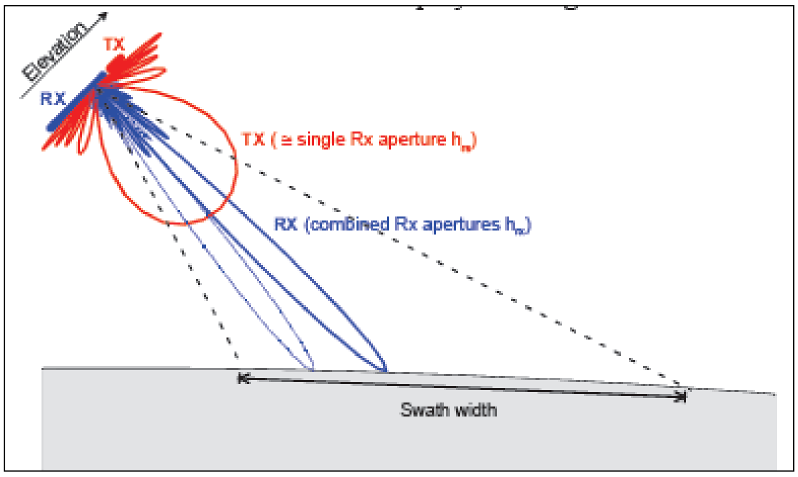

Elevation
TX
RX
TX (≥ single Rx aperture h_rx)
RX (combined Rx apertures h_rx)
Swath width

Figure 50: Scan-On-Receive (SCORE) Technique

6.2.2 Functional Requirements

The requirements defined in the following sections will be verified either by test or analysis of the demonstrator design as it will be specified on the related Test Plan Procedure document.

DRQ- 1: Digital Front-End

The digital processing architecture front-end shall match the receive antenna multi-aperture structure and subsequent receive down-conversion channels configuration. It shall receive as input signals MxN N_{bit}-digitized base-band complex (I+jQ) signals (i.e. N_{bit} bits for the I component and N_{bit} bits for the Q component) .

The digital front-end configuration cascaded with the processing architecture are schematized in next figure and coincides with the ARPA project paradigm. For the sake of simplicity MxN ADC's are represented in next figure, each sampling both the I and Q signal components.

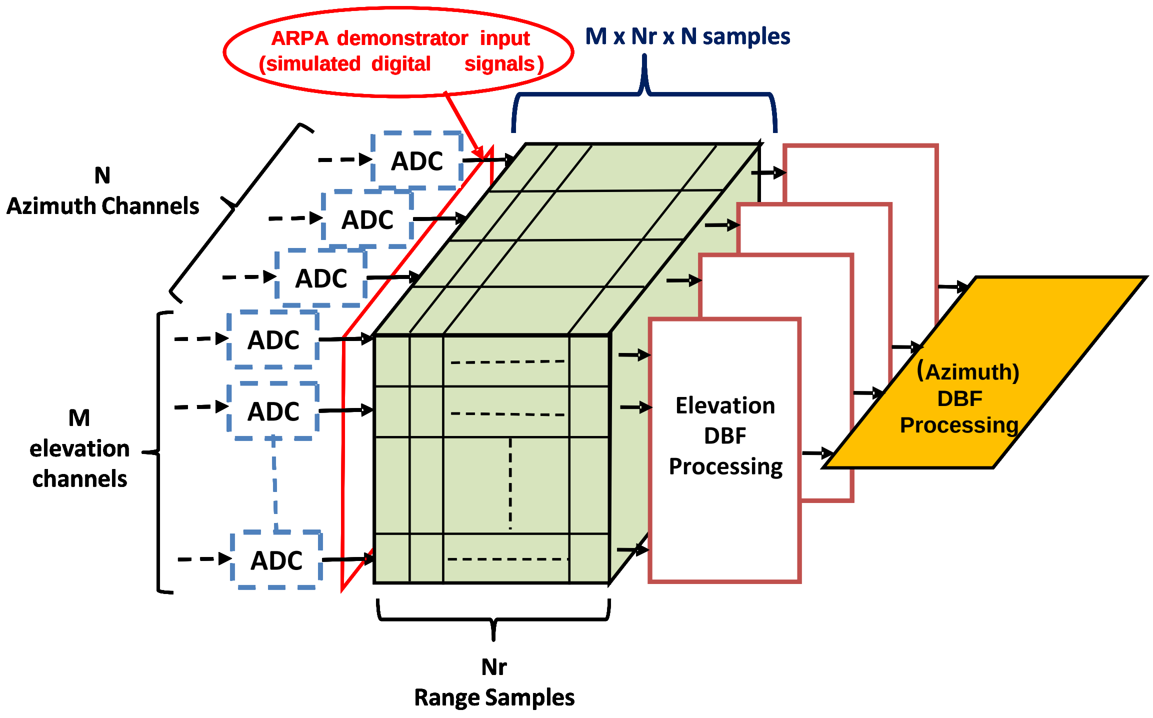


Figure 51: Digital Beam Forming (DBF) Processing Scheme for each Satellite Azimuth Position.

DRQ- 2: Input Data Characteristics

For each of the N_{az} satellite positions within the Radar synthetic aperture, the simulated backscattered signal shall be fed simultaneously to the M x N demonstrator inputs.
Each digitized signal comprises N_r samples, where N_r indicates the number of range resolution bins forming the echo signal with a *SWL* seconds duration. Accordingly, for each of the N_{az} synthetic aperture positions, MxN_rxN $2N_{bit}$ complex samples (i.e. N_{bit} bits for the I component and N_{bit} bits for the Q component) shall be available for the subsequent processing. Numbers shall be represented in Two's Complement.

DRQ- 3: Processing Flow

The processing flow shall be based on the following steps:

a. The processing shall start with the receive DBF in Elevation, implementing the SCORE technique described in the previous section. The SCORE technique will be carried out at each satellite position within the synthetic aperture. For each of the N sets of M elevation channels the same "Elevation DBF Processing" shall be carried out on the relevant MxN_r samples thus outputting a unique N_r-samples complex signal.
b. The NxN_r samples generated by the Elevation DBF Processing for each synthetic aperture position shall be stored within the EMC2 demonstrator.
c. At the end of the Radar synthetic aperture, N_{az} memory data matrices of dimension NxN_r shall be available to carry out the "Azimuth DBF Processing".
d. The Azimuth DBF Processing output shall comprise $(NxN_{az})xNr$ complex samples, which shall be parallel-to-serial converted by a proper P/S Interface and finally compressed with a "Block Adaptive Quantizer" (BAQ) algorithm.

DRQ- 4: Processing Execution Time

The whole EMC2 demonstrator processing flow shall start at the beginning of the Radar synthetic aperture and shall be completed between $2T_{AS}$ and $4T_{AS}$ seconds, where T_{AS} is the synthetic aperture duration. The effective required time shall be evaluated for the EMC2 demonstrator.

The Elevation DBF Processing, Azimuth DBF Processing, and BAQ Processing requirements are defined in detail in the following sub-sections.

6.2.2.1 Elevation Digital Beam Forming Processing Requirements

Let us consider the multi-channel receiver configuration at one of the N_{az} satellite positions within the synthetic aperture. Such a configuration corresponds to one of the M x N_r complex sample sets forming the M x Nr x N complex sample set represented in next figure. Each of the M different channels is denoted by a different *m* index value, (for *m* = 1 to M). The lower part of the figure shows the DBF processing to be carried out on these data.

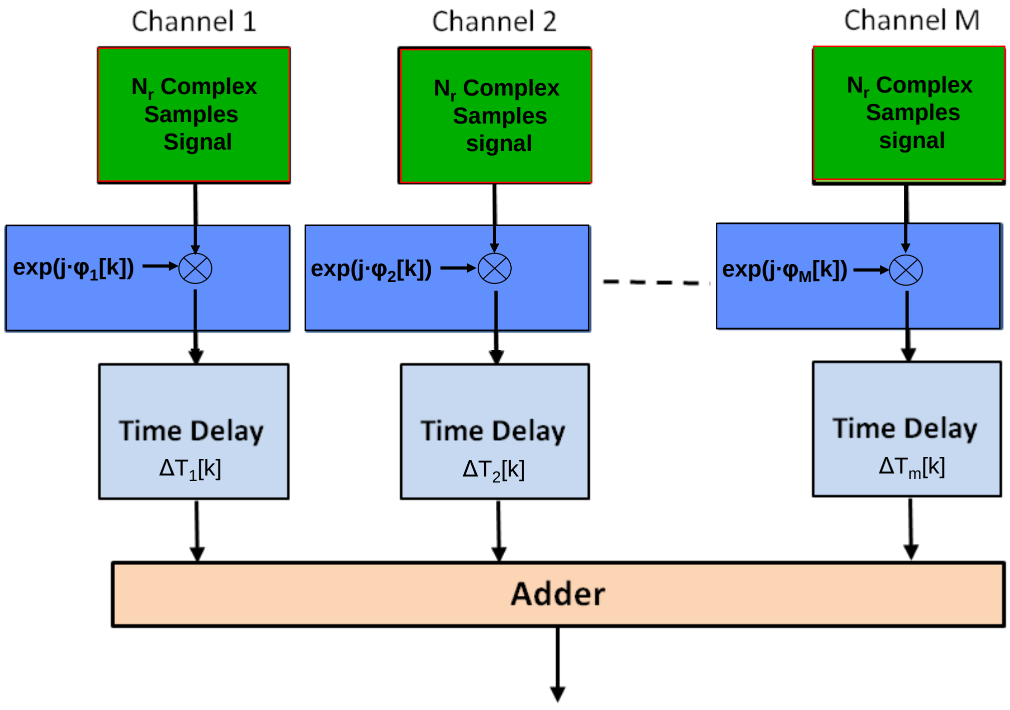


Figure 52: Elevation Digital Beam Forming (DBF) Processing Scheme

DRQ- 5: Elevation DBF Processing

The N_r-complex-samples *m*-th signal $s_m[k]$ received by the *m*-th channel , shall be processed separately through the following steps[1]:

a. For each *m*-th elevation channel a N_r sample time-variant Phase-Shift, $\exp(j\Phi_m[k])$, shall be multiplied by the $s_m[k]$ signal. The Phase-Shift shall be defined as:

$$\phi_m[k] = -\frac{4\pi}{\lambda_0} d_{el}(m-1)\sin\theta_{scan}[k] \qquad ,$$

where[2]

[1] *k* is the discrete time index related to the *digital sampling frequency* F_S *i.e. k=1,2,…,* N_r

[2] Such a Receive Beam Scanning Angle Span allows fitting the -3dB Transmit Beam Angular Aperture in elevation from Near-Range to Far-Range.

$$\theta_{scan}[k] = \left(\theta + \frac{\theta_R}{2} - \frac{\theta_P}{2}\right) - (s-1)\frac{\theta_R - \theta_P}{N_{scan} - 1}$$

and the *s index spans 1 trough N_{scan} (i.e. $s=1,2,\ldots,N_{scan}$) according to the following discrete time index set constraint*

$$\forall\, k \in [1,2,\ldots N_r] \Rightarrow s \in [1,2,\ldots N_{sacan}] \text{ s.t. } (s-1)\cdot\left\lceil \frac{N_r}{N_{scan}} \right\rceil < k \le s\cdot\left\lceil \frac{N_r}{N_{scan}} \right\rceil$$

The requirements for the parameter values necessary to compute $\Phi_m[k]$ through the previous expressions are reported in following paragraph.

b. For each *m*-th elevation channel a Time-Delay, *$\Delta T_m[k]$*, shall be applied as shown on next figure. The *m*-th Time-Delay shall be defined as:

$$\Delta T_m[k] = \frac{2}{c} d_{el}(m-1)\sin\theta_{scan}[k]$$

The parameter values required to compute *$\Delta T_m[k]$* as per the previous expressions are reported in following paragraph.

c. After executing the *(a)* and *(b)* steps for each *m*-th channel, the M channel outputs shall be summed in order to generate an unique N_r-sample output signal as shown on next figure.

DRQ- 6: Elevation DBF Parameters Computation and Storage

The values of $\Phi_m[k]$ and *$\Delta T_m[k]$* shall be computed and stored within the EMC2 demonstrator memory before the EMC2 demonstrator processing starts.

6.2.2.2 Azimuth DBF Processing Requirements

The N_{az} NxN_r data matrices generated by the Elevation DBF Processing at each satellite position shall form, at the end of the Radar Synthetic Aperture, a $N_{az}xNxN_r$ complex-samples memory tensor. Accordingly, for each range sample, the NxN_{az} slow-time samples of the signal constructed in azimuth by the synthetic aperture process shall be fed to the Azimuth DBF Processing as shown in next figure.

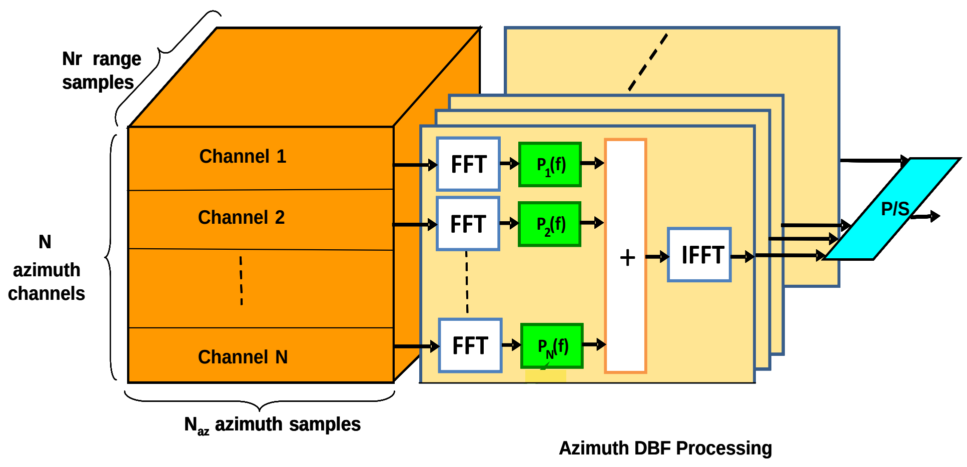


Figure 53: Azimuth DBF Processing Scheme

DRQ- 7: Azimuth DBF Processing

For each range sample, denoted by the index value *r* (for *r* =1,…,N_r), the Azimuth DBF Processing shall be carried out as per the following steps:

a. The N_{az}-complex-samples signals $s_n(t)$, (for $n=1,...,N$) shall be Fourier Transformed via a Fast Fourier Transform (FFT) processing block as shown in next figure in order to transform the signal domain from the slow-time domain, into the Doppler frequency domain, $S_n(f)$[3].

b. Each N_{az}-point Doppler Spectrum $S_n(f)$ generated at step *(a)* shall be point-wise multiplied by the *n*-th reconstruction functions $P_n(f)$ represented by the N_{az}-point subfunctions $P_{nj}(f)$ (for *j* =1,…,N) and stored as the $N\cdot N_{az}$-points function $R_n(f)$ as per the Interleaving[4] shown in next figure.

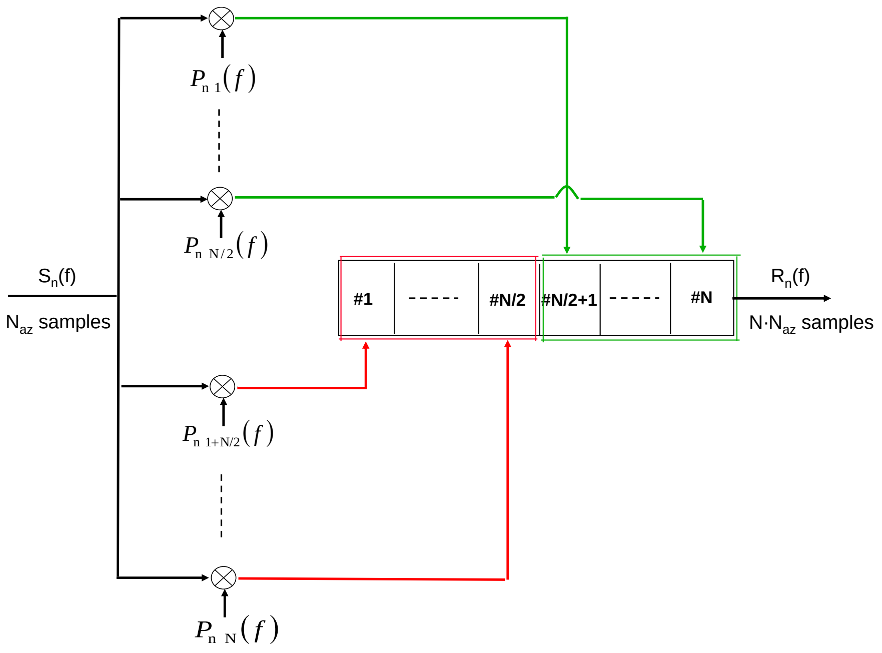


Figure 54: n-th Interleaving Block Diagram for the Azimuth DBF Processing

The N $P_n(f)$ reconstruction functions (i.e. the N^2 $P_{nj}(f)$ subfunctions) can be organized as per the following Reconstruction Matrix **P**(f):

$$P(f)=\begin{bmatrix} P_{11}(f) & P_{12}(f+PRF) & \dots & P_{1N}(f+(N-1)PRF) \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1}(f) & P_{N2}(f+PRF) & \cdots & P_{NN}(f+(N-1)PRF) \end{bmatrix}$$

Accordingly the N^2 reconstruction components, $P_{nj}(f)$, shall be computed as

$$\mathbf{P(f)} = N\cdot \mathbf{H(f)}^{-1}$$

where **H**(f) is a NxN matrix having the following form[5]:

[3] Without lack of generality the digital slow-time domain signal has been herein represented in the continuous time domain t by $s_n(t)$. Similar considerations follow for the representation of the continuous Doppler frequency f by $S_n(f)$. Proper representation in the dicrete time and frequency domains will be outlined in the forthcoming part of the section.
[4] Such an interleaving allows representing the $R_n(f)$ Doppler Spectrum from from DC to N·PRF.
[5] For the EMC2 demonstrator purposes the H(f) matrix is non-singular

$$H(f) = \begin{bmatrix} H_1(f) & \cdots & H_N(f) \\ H_1(f+PRF) & \cdots & H_N(f+PRF) \\ \vdots & \ddots & \vdots \\ H_1(f+(N-1)PRF) & \cdots & H_N(f+(N-1)PRF) \end{bmatrix}$$

The H(f) entries can be computed from[6]

$$H_n(f) \cong \exp\left[-j\frac{\pi d_{az}^2 (n-1)^2}{2\lambda_0 R_0}\right] \cdot \exp\left[-j\frac{\pi d_{az}(n-1)}{v} f\right]$$

where *R_0* is the reference slant range distance[7]

$$R_0 = (R_E + H)\cdot\cos\left(\frac{\pi}{2}-\theta\right) - \sqrt{\left[(R_E+H)\cdot\cos\left(\frac{\pi}{2}-\theta\right)\right]^2 - \left[H^2 + 2R_E H\right]}$$

while the other parameters are defined in § 6.2.2.3**.**

c. The $N\cdot N_{az}$-point functions $R_n(f)$ (for *n=1,...,N)* shall be summed in order to obtain the $N\cdot N_{az}$-point spectrum $S_{rec}(f)$ of the reconstructed signal.

d. A $N\cdot N_{az}$-point Inverse Fast Fourier Transform (IFFT) shall be carried out on the $S_{rec}(f)$ signal in order to transform the signal domain from the Doppler frequency domain into the slow-time domain thus obtaining the signal $s_r(t)$ for a given r (for $r = 1,\ldots,N_r$)[8].

e. The $N\cdot N_{az} x N_r$ data samples resulting from the overall azimuth processing shall be P/S outputted towards the subsequent BAQ processing in the following order:

```
for u = 1 to $N\cdot N_{az}$
    for r = 1 to $N_r$
        [$s_r[u]$]
    end
end
```

DRQ- 8: Azimuth DBF Parameters Computation and Storage

The values of the $P_{nj}(f)$ functions shall be computed and stored within the EMC2 demonstrator memory before the EMC2 demonstrator processing starts.

[6] The following expression results from assuming a straight line spacecraft trajectory and a Mac Laurin expansion of the received signal phase up to the 2nd order.

[7] Clearly R*o* varies between the near and far range on the oblate spheroidal Earth However the $H_n(f)$ variations as a function of R*o* are negligible. Accordingly, for the EMC2 demonstrator purposes, the following expression for R*o* is nevertheless a proper approximation considering a spherical Earth.

[8] So far the $s_r(t)$ signal was referred to according to a continuous-time domain representation for notation. The equivalent dicrete-time domain representation can be defined as $s_r[u]$ (for u = *1 to* $N\cdot N_{az}$).

6.2.2.3 EMC2 Demonstrator Parameter values

DRQ- 9: Parameter Constraints for the EMC2 Demonstrator

All the parameter values required to implement the digital signal processing techniques are reported in the following table.

Definition	Notation	Requirement
Center Wavelength	λ_0	0.031 m (X-band)
Satellite Altitude	H	580 km
Satellite Velocity	v	7560 m/s
Transmit Elevation Angle (off-boresight)	θ	35°÷55°
Transmit antenna size in azimuth	*l*	≥ 4 m
Transmit antenna size in elevation	w	≥ 0.38 m
Number of receive sub-apertures/channels in azimuth	N	≤ 2
Receive sub-aperture size in azimuth	d_{az}	≥ 4 m
Receive antenna size in azimuth	L	$N \cdot d_{az}$
Number of receive sub-apertures/channels in elevation	M	≤ 8
Receive sub-aperture size in elevation	d_{el}	≥ 0.16 m
Receive antenna size in elevation	W	$d_{el} \cdot M$
Transmit beam -3dB angular aperture in elevation	θ_R	≤ 4.6°
Receive beam angular aperture in elevation	θ_P	≥ 1.38°
Number of receive beam steering positions	N_{scan}	≥ 5
Approximate Spherical Earth Radius	R_E	6371 Km
Pulse Repetition Frequency	PRF	≤ 7000
Sampling Window Length	SWL	≤ 1500 usec
Receiver Sampling frequency	F_s	≤ 200 MHz
Analog-to-Digital Converter bit number	N_{bit}	8
Number of range samples	N_r	SWL·Fs
Radar Synthetic Aperture Time	T_{AS}	$\geq \frac{H\lambda_0}{lv\cos\theta}$
Number of transmit satellite positions within the Synthetic Aperture Time	N_{az}	$T_{AS} \cdot PRF$

Table 12: EMC2 Demonstrator Constraints

In particular the following Operative Scenario shall be outlined for the EMC2 demonstrator.

Center Wavelength	λ_0	0.031 [m]
Satellite Altitude	H	580 [km]
Satellite Velocity	v	7560 [m/s]
Transmit Elevation Angle	θ	35 [deg]
Transmit antenna size in azimuth	l	4 [m]
Transmit antenna size in elevation	w	0.38 [m]
Number of receive sub-apertures/channels in azimuth	N	2
Receive sub-aperture size in azimuth	d_{az}	4 [m]
Receive antenna size in azimuth	L	$N \cdot d_{az}$
Number of receive sub-apertures/channels in elevation	M	8
Receive sub-aperture size in elevation	d_{el}	0.19 [m]
Receive antenna size in elevation	W	$d_{el} \cdot M$
Transmit beam -3dB angular aperture in elevation	θ_R	4.6 [deg]
Receive beam angular aperture in elevation	θ_P	1.38 [deg]
Number of receive beam steering positions	N_{scan}	5
Approximate Spherical Earth Radius	R_E	6371 [Km]
Pulse Repetition Frequency	PRF	2000 [Hz]
Sampling Window Length	SWL	400 [usec]
Receiver Sampling frequency	F_s	200 [MHz]
Analog-to-Digital Converter bit number	N_{bit}	8
Number of range samples	N_r	80000
Radar Synthetic Aperture Time	T_{AS}	$N_{az}/PRF \approx 8.19$ [s]
Number of transmit satellite positions within the Synthetic Aperture Time	N_{az}	2^{14}

Table 13: Operative Scenario

Without lack of generality for the EMC2 project, the TX/RX1/RX2 Antenna geometrical relationships shown in the figure reported hereafter will be taken into account for computing the 2x2 Reconstruction Matrix P(f).

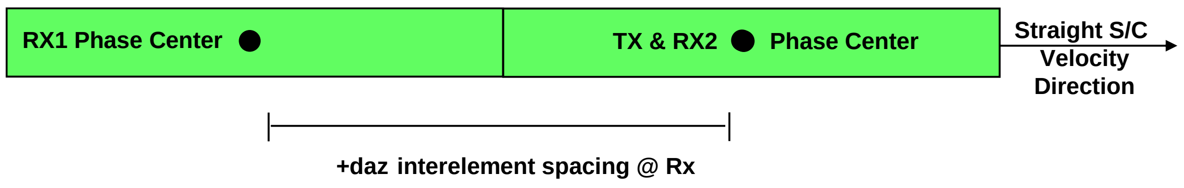

RX1 Phase Center
TX & RX2 Phase Center
Straight S/C Velocity Direction
+daz interelement spacing @ Rx

Figure 55: Tx/Rx Antenna Geometrical Relationships

Accordingly the $P_{nj}(f)$ subfunctions (for i,j =1,…,2) are[9]

[9] For the sake of completeness the subfunctions $P_{nj}(f)$ frequency domain in Hz is [-PRF , 0) for $P_{11}(f)$ and $P_{21}(f)$ while [0 , PRF) for $P_{12}(f)$ and $P_{22}(f)$.

$$P_{11}(f)=\frac{2}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]\cdot e^{j\cdot\left(-\frac{2\pi\cdot d_{az}}{2\cdot v}\right)\cdot f}} \qquad P_{12}(f)=-\frac{2}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]\cdot e^{j\cdot\left(-\frac{2\pi\cdot d_{az}}{2\cdot v}\right)\cdot(f-PRF)}}$$

$$P_{21}(f)=-\frac{2\cdot e^{j\varphi_2}}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]} \qquad P_{22}(f)=\frac{2\cdot e^{j\varphi_1}}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]}$$

where

$$\varphi_1=-\pi\cdot\frac{(d_{az})^2}{2\cdot\lambda_0\cdot R_0} \qquad \varphi_2=-\pi\cdot\left[\frac{(d_{az})^2}{2\cdot\lambda_0\cdot R_0}+\frac{d_{az}}{v}\cdot\mathrm{PRF}\right]$$

On the complex z-plane, the N_{az} *radix-2-FFT* points are equally distributed around the unit circle and therefore

$$f_k=\frac{PRF}{N_{az}}\cdot k$$

where *k* is the discrete index. Finally for the EMC2 demonstrator the N_{az} points $P_{nj}[k]$ subfunctions (for i,*j* =1,…,2) appear as

$$P_{11}[k]\Big|_{k=-N_{az},-N_{az}+1,-N_{az}+2,\dots,-1}=\frac{2}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]\cdot e^{j\cdot\left(-\frac{2\pi\cdot d_{az}}{2\cdot v}\right)\cdot\left(\frac{PRF}{N_{az}}\cdot k\right)}}$$

$$P_{12}[k]\Big|_{k=0,1,2,\dots,N_{az}-1}=-\frac{2}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]\cdot e^{j\cdot\left(-\frac{2\pi\cdot d_{az}}{2\cdot v}\right)\cdot\left(\frac{PRF}{N_{az}}\cdot k-PRF\right)}}$$

$$P_{21}[k]\Big|_{k=-N_{az},-N_{az}+1,-N_{az}+2,\dots,-1}=-\frac{2\cdot e^{j\varphi_2}}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]}$$

$$P_{22}[k]\Big|_{k=0,1,2,\dots,N_{az}-1}=\frac{2\cdot e^{j\varphi_1}}{\left[e^{j\varphi_1}-e^{j\varphi_2}\right]}$$

6.2.2.4 Block Adaptive Quantizer (BAQ) Processing Requirements

6.2.2.4.1 General Description

The Block Adaptive Quantizer (BAQ) algorithm shall be applied to the Azimuth DBF Processing serial output in order to compress the 8-bits complex digital samples (by reducing the number of bits needed to represent each sample of the I and Q components) with no degradation for the SAR performance.

The BAQ algorithm is based on two steps:

- Normalization of the 8-bits I/Q samples at the Azimuth DBF Processing serial output.

- Adaptive Re-Quantization of the normalized I/Q samples with a reduced number of bits.

The samples normalization is computed on blocks of 128 samples and is based on an estimate of each block power. A block power is computed by summing the absolute value of the I and Q samples within the block.

For every 128 samples block, the Adaptive Re-Quantization is based on selecting the dynamic range of each 8 bits complex sample by means of optimum converters, in the following also referred to as “quantizers”, whose thresholds are related to the block-power-measurement. The thresholds of such quantizers shall be pre-computed and stored within the EMC2 demonstrator memory.

The BAQ Compression Ratio is defined as the ratio between the number of bits associated to each sample before compression (8 bits) and after compression (Z bits).

In the following sections, the algorithm is specified for the following compression ratios :
8:4, 8:3, 8:2 and 8:1 (i.e. 8:Z with Z=4, 3, 2, 1).

For the 8:1 Compression Ratio, the BAQ extracts (from the Azimuth DBF Processing serial output) as a 1 bit complex output only the sign of the I and Q samples. In this case the estimate of the sample-block power shall be computed solely to pinpoint the signal power information.

6.2.2.4.2 BAQ Characteristics

DRQ- 10: Implemented Compression Ratios

Four different compression ratios shall be implemented: 8:4, 8:3, 8:2 and 8:1.

DRQ- 11: Input Dynamic Range

The dynamic range of the 8-bits Azimuth DBF Processing serial complex output (i.e. 8 bits for the I component and 8 bits for Q component) shall be spanned by 16 Quantizers Schemes (QS) for each of the 4 compression ratios (i.e. Z=4, 3, 2, 1).

DRQ- 12: Quantizers Characteristics

For each Compression Ratio Z, each of the 16 BAQ Quantizer Schemes (QS) is defined by a set of 15 “Quantizer Selection Thresholds” (QST) and a set of “Quantizer Thresholds” (QT).

The number of Quantizer Thresholds QT for each of the 16 BAQ Quantizer Schemes QS, depends on the Compression Ratio Z and is given by:

$$M = 2^{Z-1} - 1$$

Next Table reports, for each Compression Ratio, the number of QTs for each Compression Ratio and the total number of QTs.

Compression Ratio (Z)	Number of QTs for each QS (M)	Total Number of QTs (16·M)
8:4	7	112
8:3	3	48
8:2	1	16
8:1	0	0

Table 14: Quantizer Thresholds

DRQ- 13: Storage Requirement

The total number of BAQ parameter values to be stored for each Compression Ratio, considering both QSTs and QTs is reported in Next Table:

Compression Ratio (Z)	Total Number of BAQ Parameter Values
8:4	112+15
8:3	48+15
8:2	16+15
8:1	15

Table 15: Number of BAQ Parameter Values to be stored for each Compression Ratio

DRQ- 14: Input Data Coding

The BAQ input data shall be coded into 8 bits Sign & Magnitude (i.e. 7 bits for the magnitude and 1 bit for the sign) for each I and Q sample, as shown in the following table.

D7	D6	D5	D4	D3	D2	D1	D0	value
0	1	1	1	1	1	1	1	+127
•	•	•	•	•	•	•	•	
•	•	•	•	•	•	•	•	
0	0	0	0	0	0	0	1	+1
0	0	0	0	0	0	0	0	+0
1	0	0	0	0	0	0	0	-0
1	0	0	0	0	0	0	1	-1
•	•	•	•	•	•	•	•	
•	•	•	•	•	•	•	•	
1	1	1	1	1	1	1	1	-127

Table 16: Sign & Magnitude Coding

The N_{bit} bits complex samples at the P/S Interface output shown in next figure shall be transformed from a Two's Complement representation into a Sign & Magnitude representation according to the following mapping[10]:

for each Two's Complement number whose decimal value is X the corresponding Sign & Magnitude number whose decimal value is Y will be Y=X if X>=0 else Y=X+1.

DRQ- 15: Sign Coding

The sign shall be coded as a 0 for positive numbers and as a 1 for negative ones.

DRQ- 16: Threshold Encoding

The QTs shall be coded as unsigned 7 bit integers, while the QSTs shall be coded as 15 bit unsigned integers.

DRQ- 17: Upper Bound Threshold

[10] Such a mapping results in inverting the 7 Least Significant Bits (LSB) of negative two's complement numbers i.e. those whose Most Significant Bit (MSB) is 1. Positive two's complement numbers (i.e. those whose MSB is 0) are left unaltered.

For each Compression Ratio, each QS shall have an implicit upper-bound QT_{M+1} set to 127.

6.2.2.4.3 Power Estimation

The *i*-th POW_i power estimate shall be computed on the *i*-th 128-samples block, by summing the absolute values of the I and Q samples within the block.

DRQ- 18: BAQ Block Segmentation

The I and Q samples fed to the BAQ Processing input shall be divided in blocks of 128 samples each.

DRQ- 19: Power Estimation

For each *i*-th block the SQ_i power estimate shall be computed as follows:

$$POW_i = \sum_{n=1}^{128} \left(|I_n| + |Q_n| \right) \Big|_i$$

and shall be represented as a 15 bits unsigned integer.

In case the total number of BAQ Processing input samples is not a multiple of 128, the last block called "Residual Block", will be shorter than 128 samples.

DRQ- 20: Residual Block Power Estimation

No Power estimate shall be performed for the Residual Block.

6.2.2.4.4 Quantizer selection

For each Compression Ratio, the *k*-th QS_k (for k=1,2,…,16) shall be selected by comparing the computed Power POW_i for the i-th block with the set of 15 QSTs..

DRQ- 21: QS Selection

For each Compression Ratio, the QS_k to be selected shall be the largest one for which:

$$POW_i \leq QST_k$$

for k=1,…15. In case

$$POW_i > QST_{15}$$

the QS to be selected shall be QS_{16}.

The *k*-index of the selected QS_k to be used for a Residual Block shall be the one used for the previous block.

6.2.2.4.5 Samples Coding

Each I_n and Q_n sample (for n=1,…,128) within each i-th block shall be coded as per the following procedures:

DRQ- 22: I_n and Q_n Sample Sign Extraction

The sign of each I_n and Q_n sample shall be extracted from the most significant bit of the sample Sign & Magnitude representation

DRQ- 23: $|I_n|$ and $|Q_n|$ Sample Coding

For all Compression Ratios except 8:1 each $|I_n|$ and $|Q_n|$ sample shall be compared to the QT_m values. The *m* index (for m=1,…, M+1) to be selected shall be the largest one for which:

$$|I_n|_i \leq QT_m$$

$$|Q_n|_i \leq QT_m$$

For all Compression Ratios except 8:1 the output $|I_n|$ and $|Q_n|$ sample coding shall be m-1.

DRQ- 24: I_n and Q_n Sample Coding

a. For the 8:4 Compression Ratio, the output binary coding shall be the following:

m	B4	B3	B2	B1
1	SIGN	0	0	0
2	SIGN	0	0	1
3	SIGN	0	1	0
4	SIGN	0	1	1
5	SIGN	1	0	0
6	SIGN	1	0	1
7	SIGN	1	1	0
8	SIGN	1	1	1

Table 17: 8:4 Compression Ratio Binary Coding

b. For the 8:3 Compression Ratio, the output binary coding shall be the following:

M	B4	B2	B1
1	SIGN	0	0
2	SIGN	0	1
3	SIGN	1	0
4	SIGN	1	1

Table 18: 8:3 Compression Ratio Binary Coding

c. For the 8:2 Compression Ratio, the output binary coding shall be the following:

m	B2	B1
1	SIGN	0
2	SIGN	1

Table 19: 8:2 Compression Ratio Binary Coding

d. For the 8:1 Compression Ratio, the output binary coding shall be the sign as per DRQ-22.

6.2.2.4.6 QSTs Values

The values for the QSTs for each Compression Ratio are reported in the following tables.

DRQ- 25: Quantizer Selection Thresholds

a. For the 8:4 Compression Ratio the 15 QSTs shall be:

QST15	**QST14**	**QST13**	**QST12**	**QST11**	**QST10**	**QST9**	**QST8**	**QST7**	**QST 6**	**QST 5**	**QS T4**	**QST3**	**QST2**	**QST 1**
9593	7978	6617	5483	4539	3754	3101	2590	2081	1753	1384	1220	995	807	678

b. For the 8:3 Compression Ratio, the 15 QSTs shall be:

QST15	**QST14**	**QST13**	**QST12**	**QST11**	**QST10**	**QST9**	**QST8**	**QST7**	**QST 6**	**QST 5**	**QS T4**	**QST3**	**QST2**	**QST 1**
12599	9981	7794	6060	4703	3644	2817	2172	1669	1252	969	745	544	408	296

c. For the 8:2 Compression Ratio the 15 QSTs shall be:

QST15	**QST14**	**QST13**	**QST12**	**QST11**	**QST10**	**QST9**	**QST8**	**QST7**	**QST 6**	**QST 5**	**QS T4**	**QST3**	**QST2**	**QST 1**
18841	14996	11213	8071	5747	4080	2886	2031	1419	957	687	433	325	188	102

d. For the 8:1 Compression Ratio the 15 QSTs shall be:

QST15	**QST14**	**QST13**	**QST12**	**QST11**	**QST10**	**QST9**	**QST8**	**QST7**	**QST 6**	**QST 5**	**QS T4**	**QST3**	**QST2**	**QST 1**
23048	19156	14590	10208	6854	4566	3028	1994	1300	834	522	313	175	86	31

6.2.2.4.7 QTs Values

The values for the quantizer thresholds QTs for each Compression Ratio are reported in the following tables.

DRQ- 26: Quantizer Thresholds

a. For the 8:4 Compression Ratio, the 16x7 QTs shall be:

QS_k	**QT1**	**QT2**	**QT3**	**QT4**	**QT5**	**QT6**	**QT7**
16	13	27	41	57	75	96	126
15	10	22	34	47	62	79	104
14	8	18	28	39	51	65	86
13	7	15	23	32	42	54	71
12	5	12	19	26	34	44	58
11	4	10	15	21	28	37	48
10	3	8	12	18	23	30	39
9	3	6	10	14	19	25	32
8	2	5	8	12	16	20	27
7	1	4	7	9	13	17	22
6	1	3	5	8	10	14	18
5	1	2	4	6	8	11	15
4	0	2	3	5	7	9	12
3	0	1	3	4	5	7	10
2	0	1	2	3	4	6	8
1	0	1	1	2	3	5	6

b. For the 8:3 Compression Ratio, the 16x3 QTs shall be the following:

QS_k	**QT1**	**QT2**	**QT3**
16	35	75	126
15	27	58	98
14	21	45	76
13	16	35	59
12	12	27	46
11	10	21	36
10	7	16	28
9	5	12	21
8	4	9	16
7	3	7	13
6	2	5	10
5	1	4	7
4	1	3	5
3	0	2	4
2	0	1	3
1	0	1	2

c. For the 8:2 Compression Ratio, the 16x1 QTs shall be:

QS_k	**QT1**
16	126
15	90
14	64
13	45
12	32
11	23
10	16
9	11
8	8
7	5
6	3
5	2
4	1
3	1
2	0
1	0

6.3 Conclusions for Radar Payload Applications

From a detailed Survey of HPPM Components for the HPEC Architecture Study Case it is possible to:

1) Exclude all FPGAs embedded solutions, because of low performances and poor radiation data (Virtex-4 is the only one available at QV level).
Also open SPARC looks not having reached a level of diffusion such to encourage to go in that direction.

2) confirm the interesting on ARM, or at least some of the Cores, but they need to licensing and then find a hardware implementation makes them less attractive, although they have a very large popularity. Also components embedding ARM cores (Freescale in particular could be attractive), are very specialized, embedding several functions not relevant for space application.

3) The Freescale/e2v PowerPC family looks today the most attractive way to follow because of:

a) Thales Alenia Space is already cooperating with e2v, for PC7448
b) e2v has the long term agreement with Freescale, to ensure long term availability of the selected product
c) In addition also Thales, and in particular Thales Avionics, have close relations with Freescale and e2v. This in particular includes NDA between Thales and Freescale.

For 8 or more cores the initial choice shall be P4080, to evolve to T4160 or T4240.

7. References

[1] Cobham Gaisler, Quad Core LEON4 SPARC V8 Processor, LEON4-N2X Data Sheet and User’s Manual, datasheet, 2014.

[2] Cobham Gaisler, GR-CPCI-XC4V Development Board User Manual, 2013.

[3] SYSGO AG, *PikeOS Fundamentals*, datasheet, 2009.

[4] FentISS, "XtratuM Hypervisor for Multicore LEON3/4", User Manual, 2013.

[5] CCSDS 122.0-B-1, Image Data Compression, Blue Book Issue 1, 11/2005