Design, development, and implementation of a cFS, RTEMS, and LEON3 platform.

Jesús Zurera* Miguel A. de Miguel* Hugo Valente* Ángel G. Pérez* Alejandro Alonso* Juan Zamorano* Juan A. de la Puente*

**Universidad Politécnica de Madrid, ETSI Telecomunicación, Ciudad Universitaria, Madrid, Spain*

e-mail: j.zurera@upm.es

Abstract: As interest in space missions and the services they provide grows, the need to design, develop, and implement new platforms that can support the necessities of these services becomes more evident. To this end, the implementation of one of the most used software architectures in space missions, the core Flight System (NASA), to work in the Real Time Operative System RTEMS and the 32-bit LEON3/SPARC V8 processor. In order to test this platform, some applications which prove the main functionalities of the cFS software framework have been developed such as communication between components through a publish subscribe pattern, events, and data storage. These applications have been characterized and modeled using the AADL language, thus allowing the creation by a non-specialized user, through auto-code, of new custom applications.

Keywords: embedded software, real-time systems, cFS, RTEMS, LEON3, MBSE, AADL.

1 INTRODUCTION

This document introduces a platform for the execution of components, and blocks of software, auto-generated from a model-based tool-suite. The component behavior and relations are determined by a set of requirements which were already presented in (Zurera, 2021); they define the main characteristics a modelling language must have. The scope of this document is framed under the AURORA project. The AURORA project has the objective of providing a European tool suite for the process of development and validation of a critical Auto-coded Flight software product in the Space domain reducing the production time and cost of satellite software development.

These components must be tested in updated interoperable software architectures for Flight Software. Research among the main software architectures currently in the market has been carried out; solutions explored are AUTOSAR (AUTomotive Open System ARchitecture), the European Space Agency (Space AVionics Open Interface aRchitecture) or the Core Flight System (NASA).

The core Flight System (cFS) platform is a 3-layered flight software architecture developed by NASA Goddard Space Flight Center (GSFC). It provides an application runtime environment independent from both operating system and the hardware. This independence between layers is achieved through the modularization and creation of APIs. A first layer of applications or components that can be included independently depending on the needs of the project. A second layer that forms the core of the software, core Flight Executive (cFE), which provides the main functionalities such as messaging, events, component management, etc. Finally, a layer that abstracts the previous ones from the operating system and the hardware used, these are Operative System Abstraction Layer and Platform Support package respectively. (NASA, 2021).

These characteristics, presented in the previous paragraph, make CFS suitable for implementation as the core software of the execution platform because its core layer, cFE, meets most requirements that were set for a compliant modelling language. The objective of this platform is to test self-generated and correct-by-construction components through code generation tools and modeling languages such as AADL. These components or applications (interchangeable terms in the cFS bibliography) will be designed to demonstrate the functionality of the platform. This type of approach is based on Model-Based Systems Engineering (MBSE). (Feiler & Gluch, 2012)

RTEMS (Real-Time Executive for Multiprocessor Systems) is an Open-Source Real-Time Operating System that is available for a wide range of processor families and boards. SPARC V8 processors are supported, and it is available for several computer systems based on ERC32, LEON2, LEON3, and LEON4 processors. It is widely used in the aerospace domain and has a POSIX interface that makes RTEMS a good choice for building a platform to execute cFS applications.

We have designed a specific On-Board Computer (OBC) as a hardware platform. This hardware platform includes a LEON3 processor that has SPARC v8 architecture. (Gaisler Reaearch, 2012) (SPARC International, 1992) It also includes 4MB of RAM memory and 2MB of non-volatile EEPROM type memory to store the configuration and telemetry data for later sending to the ground station. The computer also includes analog and digital input and output interfaces for interaction with the sensors, as well as RS-232 and RS-422 interfaces for the radio, debug, and test connections, and some of the experiments. (de la Puente, Zamorano, Alonso, & Brosnan, 2012)

The software development environment is based on a LEON/ERC32 RTEMS Cross Compilation System (RCC). For this platform, the RCC 1.3.1 has been used to produce the LEON3 executables. RCC is a multi-platform development system based on the GNU family of freely available tools with additional tools developed by RTEMS Community and Cobham Gaisler. RCC allows cross-compilation of RTEMS C/C++ applications for LEON4, LEON3, LEON2 and ERC32. Using the GDB debugger, it is possible to perform source-level symbolic debugging, either on the TSIM simulator or using real target hardware via GRMON. (Cobham Advanced Electronic Solutions, 2022)

2 EXECUTION PLATFORM AND DEVELOPMENT ENVIRONMENT

To create a platform with the characteristics and components that have been described in the introduction, not only the independent software and hardware blocks that compose it are needed, but also the tools and modifications that allow them to be interconnected and relate them.

This section presents the description and changes carried out in the components and tools that are connected to each other to form the platform.

2.1 RCC

The Cross Compilation System provides support for generation an elf executable for the LEON3 OBC that includes cFS, RTEMS 5.1 C code, and software application. RCC is installed in a Linux platform and comes with the RTEMS 5.1 kernel source code included. Although RCC includes RTEMS, we use the RTEMS version included in the cFS compilation toolchain. These are known as the cross-compiler directories and include: i) C compiler *gcc*, ii) C++ compiler *g++*, iii) GNU linker *ld*, iv) Cross-assemble *as*, v) *strip*, a utility to remove symbols, vi) *nm*, utility to print symbol table, vii) Library archiver *ar*, viii) *objdump* to drop various part of executables, ix) *objcopy* to convert between different binary formats among other options and libraries. This set of basic tools gives support for the construction of software development environment.

2.2 cFS

As presented in the Introduction, one of the main features of cFS is its ability to be structured in independent layers. This allows the modification of the different parts of cFS without affecting the rest, thus being independent the development of the applications, the interface with the operating system, OSAL, and the interface with the hardware used, PSP.

As we approach the development of this platform, in the late 2021, cFS is ready to be implemented in a variety of different platforms. From the GitHub platform (NASA OSAL, 2022) (NASA PSP, 2022) these platforms are pc-Linux, pc-RTEMS and mcp750-VxWorks. The objective here is to create a new LEON3-RTEMS platform adding new OS abstraction and support packages. The architecture of the platform is presented in Figure 1 - Platform architecture.

Thus, the additions to cFS are modified from the already existing Operative System Abstraction for RTEMS and a new platform support package for LEON3. We have integrated this new version of cFS into the software development environment. This new configuration is defined in a toolchain file selected at compilation time. The most important

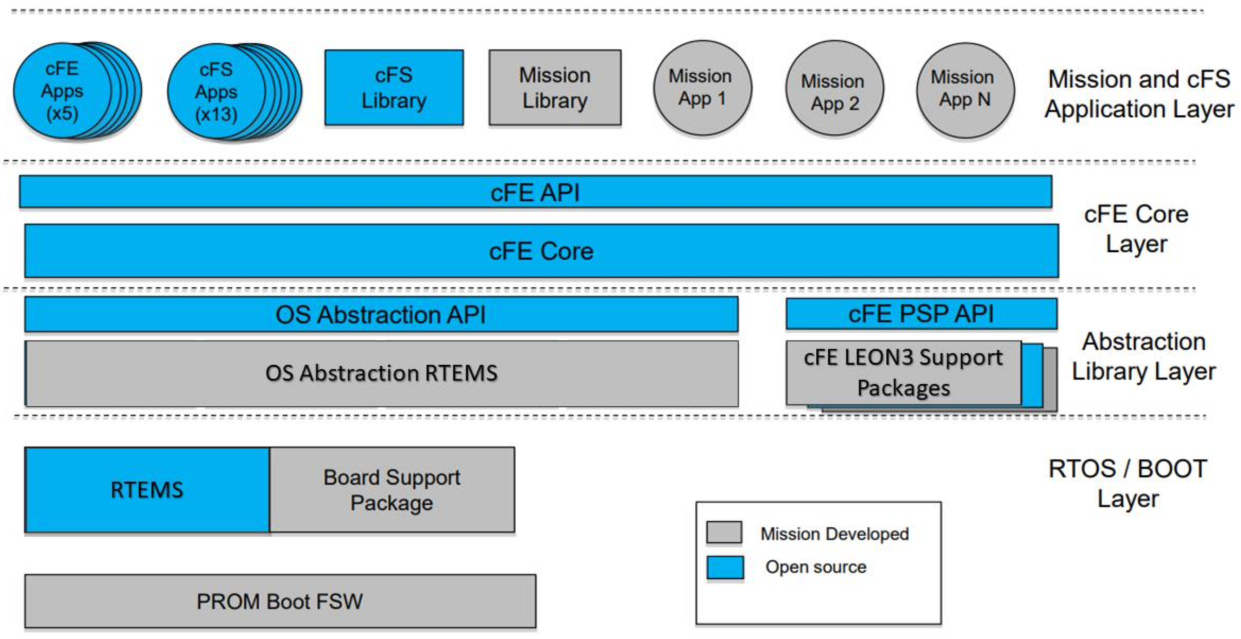

Figure 1 - Platform architecture. Figure adapted from NASA (2021)

parameters configured in this file are: i) cross compiler directories, ii) BSP, iii) compilation flags and options, iv) libraries for RTEMS, v) selection of PSP, BSP and OS type, and vi) configuration of relocation addresses for the OBC.

Operative System Abstraction Layer

The goal of the operating system abstraction layer is to create portable and reusable embedded system software in real time. All cFS software is produced to be implemented on different operating systems (OS) with appropriate specific configuration, and the same embedded software must be compiled and run on different platforms, from spacecraft computer systems to desktop PCs. This layer isolates the upper layers of the Real Time Operating System and thus allows cFS to be implemented in OS such as RTEMS, VxWorks or Linux, which is not RTOS but is suitable for developing and testing processes. (Zurera, 2021). The OS APIs are broken up into core, file system, loader, network, and timer APIs.

From the source code of OSAL the platform is launched. A leon3-rtems configuration is created and the Init task of RTEMS is declared. The configuration file is based on the pc-RTEMS one, changing, and adapting the parameters that need it. On a first BSP configuration different resources are setup like root file system, the mount point for general propose file system and the shell init, others are dismissed due to being unused.

Platform Support Package

The PSP configures the interface with the hardware used in the project. This software library contains all the information needed to suit cFS in a determined processor card, which will be different in each development. Its functions include: i) start-up code, ii) processor card reset functions, iii) EEPROM and memory read, write, copy, and protection functions, iv) timer functions, and v) exception handler functions. The PSP also provides an API to interact abstractly with hardware computing system or hardware devices such as memory, I/O ports, and non-volatile memory.

Herein lies one of the main changes from the native cFS configurations. The loading of the application and the startup script cannot be done dynamically, once already at execution time, but rather statically, adding the symbol files statically at compilation time. This is so due to the configuration of the RTEMS operating system which, despite having a dynamic loader, does not recommend its use as a real-time response to an event: "The RTEMS developers do not see dynamically loading of code as a real-time activity. A system should not respond to real-time external events by loading code. The loading of code should happen before a system is considered available, and the activity the system is experiencing is low and stable." (RTEMS, 2022)

cFS does not have, at the time of development of this platform, a method to introduce the start script statically as discussed in one of its GitHub forums. (NASA, 2022)

The solution has been to introduce a file through the code itself using the EMBEDED_LIST in the target.cmake file and copy it directly into RAM. Bind then this memory address by means of a method belonging to OSAL which allows creating a fixed mapping between an existing directory and a virtual OSAL mount point; it is intended to be called by the PSP/BSP prior to starting the application and is registered at runtime.

By modifying the aforementioned file, targets.cmake, a list of applications and their corresponding symbols can also be

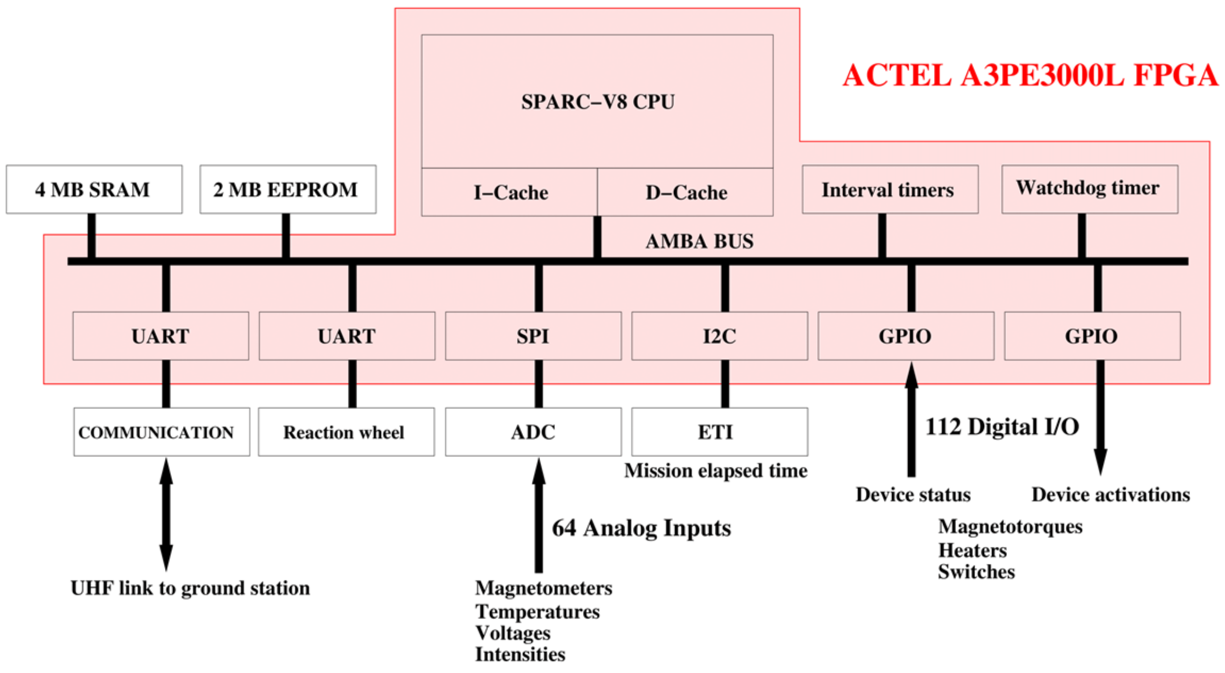

Figure 2 - LEON3 OBC architecture

statically introduced to be loaded by cFE during component initialization.

2.3 *LEON3*

The On-board Computer was developed from the VHDL GRLIB IP cores library, and it is the flight version of the UPMSat-2 microsatellite that was launched on September 3, 2020, on the flight VV16 of the Vega rocket. This library contains the IP core of the 32-bit LEON3 processor along with AMBA system bus controllers, memory controllers, as well as auxiliary bus controllers and peripheral devices. The library is available in VHDL source code and is distributed under the GNU GPL license. Therefore, it was used to develop the OBC with an adequate configuration and synthesis on an FPGA.

Figure 2 - LEON3 OBC architecture shows the structure of the OBC that was synthesized in an ACTEL (now Microsemi) A3PE3000L FPGA. It includes a LEON3 processor with data and instruction caches, one AMBA bus hierarchy together with serial interfaces including RS-232, RS-422, SPI, I2C, and 112 digital I/O signals. Magnetometers and other analog signals are read through analog input channels. The actuating signals to magnetorquers are the only analog outputs of the OBC. To simplify the hardware design and avoid the inclusion of Digital to Analog Converters (DAC) and amplifiers, digital outputs and H-bridges driven by pulse-width modulation (PWM) are used to produce the required currents.

3 PLATFORM TESTING

A set of several applications have been developed to prove the main characteristics of cFS and check if these are met in for the new execution platform implemented in this project. These applications can be divided by their objectives being the following:

- Asynchronous communication between components through the publish and subscribe methodologies.
- Asynchronous event notification to the cFS core, cFE.
- Persistent data store capabilities.
- Integration of the different functionalities in one single component.

To verify the correct functioning of the platform, the results have been compared between three different types of executions.

The first is the execution of these applications on the native cFS platform; this is on a Linux desktop PC, which, despite not being a real-time operating system, can be used as a development environment for testing. The second is to use the TSIM3 simulator for a LEON3 SPARC board, a tool provided by Cobham Gaisler. This tool has been very useful in not having to load the executable compiled by RCC directly into the OBC for each test. Lastly, using directly GRMON3, a debugging tool also provided by Cobham Gaisler, to load the platform executable file and run it in the OBC.

Comparing the operation of the platform and the execution of the different applications developed to exemplify its main characteristics, we can verify that the behavior is the desired one. Execution time measurements are made on the real OBC model.

4 APLICATION AADL MODELS

In Figure 3 - Components model, the Attitude Determination and Control System (ADCS) of the UPM Sat 2 is represented. It consists of a set of sensors (magnetometers) that read the Earth’s magnetic field, pass it to the ADCS which calculates the necessary corrections to be made based on the average of the readings and then sends the corrections to the actuator (magnetorquer) and to a storage component.

This model shows how typical capabilities that can be found in cFS such as the N to 1 and 1 to M communication patterns and persistency can be represented in Space Creator, a modelling framework that relies on AADL elements for code generation.

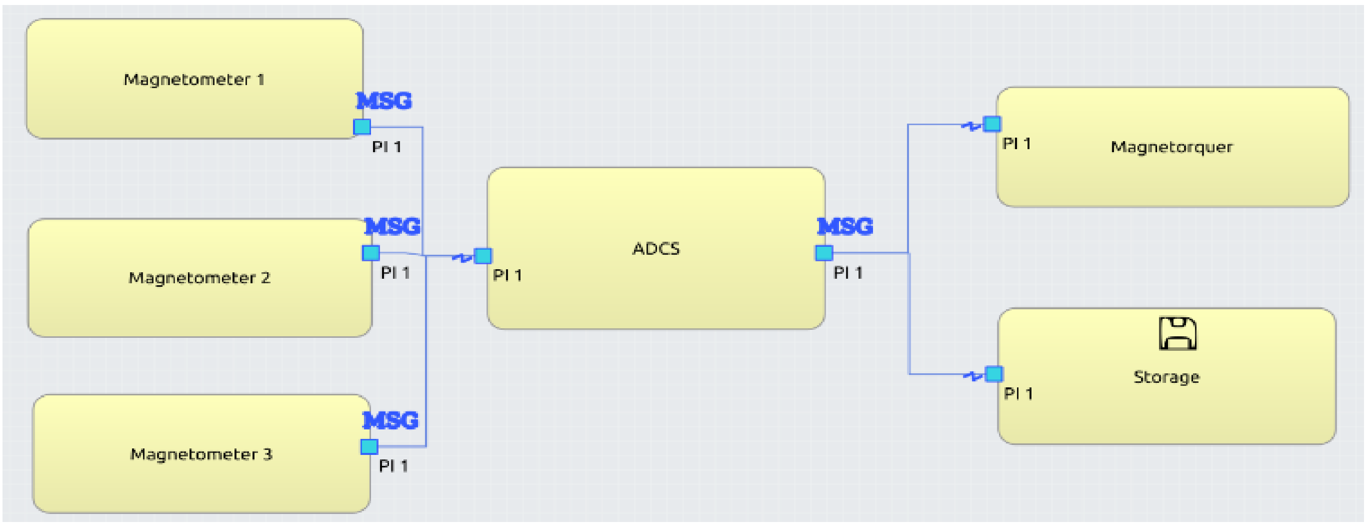

Figure 3 - Components model

The generated code consists of application specific files that are generated for each component based on the properties of the models and of global configuration files that specify the startup information required to load the component on startup. The total number of generated files for this simple project was of 62 (2 global files + 10 application files per component).

When considering the number of files generated, combined with the fact that the generated code supports patterns such as N to M, persistence, and can be executed directly on the platform without manual coding, the potential of MBSE becomes clear.. MBSE increases the development automation level by reducing the amount of repetitive and error-prone tasks, which means the developers can easily reuse solutions and instead focus on the project-specific details.

5 CONCLUSIONS AND FUTURE WORK

A platform has been achieved that adapts the versatile, reusable, and open source cFS on a real-time operating system also open source and widely used in embedded systems such as RTEMS to a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture LEON3. This is a milestone that opens many doors for us and allows possible future implementations of satellite platforms with this architecture, among others, for a future UPM-Sat3 satellite.

Finally, note how this platform allows the testing of self-generated applications using models, thus being a very useful tool for verifying the operation of modeling languages, allowing changes to be tested in these languages or even the creation of new ones. Expanding the functionalities of the models created, it will be possible to support a greater number of applications for flight software. This will allow non-expert users to create and tailor applications for their specific uses without the need to delve into the code that underlies them.

At present, two strongly correlated functionalities are being developed: component management and fault detection. The objective is to create a fault detection algorithm with notification capacity that allows the system starting, stopping, suspending, and resuming behaviors. This will allow the system to deal with faulty components without compromising overall system performance, a crucial feature in critical flight software.

Condensing all the capabilities that a flight software must present in the implemented platform will allow it to expand its use to a greater number of applications and uses in the future.

ACKNOWLEDGEMENTS

The work described in this paper has been developed within the European project AURORA (101004291) and the project OAPES-CM (Y2020/NMT-6427). We would like to acknowledge the financial support of the European Union's H2020 R+I programme, and the Comunidad de Madrid (Spain) Proyectos Sinérgicos de I+D plan, as well as the collaboration with partners in both projects.

REFERENCES

Cobham Advanced Electronic Solutions. (2022). Retrieved from LEON/ERC32 RTEMS Cross Compilation System: https://www.gaisler.com/index.php/products?option=com_content&task=view&id=150

de la Puente, J., Zamorano, J., Alonso, A., & Brosnan, D. (2012). A real-time computer control platform for an experimental satellite. *Jornadas de Tiempo Real — JTR*.

Feiler, P., & Gluch, D. (2012). Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language. *Addison-Wesley Professional*.

Gaisler Reaearch. (2012). *LEON3 - High-performance SPARC V8 32-bit Processor.* Obtenido de GRLIB IP Core User's Manual.

NASA. (2021, May 13). *cFS Background and Overview*. Retrieved from https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf

NASA. (2022). *cFS Discussions*. Retrieved from https://github.com/nasa/cFS/discussions/314

NASA cFS. (2022). *cFS*. Retrieved from https://github.com/nasa/cfs

NASA OSAL. (2022). *OSAL*. Retrieved from https://github.com/nasa/osal

NASA PSP. (2022). *PSP*. Retrieved from https://github.com/nasa/psp

RTEMS. (2022). *Dynamic Loader*. Retrieved from https://docs.rtems.org/branches/master/user/exe/loader.html

SPARC International. (1992). *The SPARC architecture manual: Version 8.*

Zurera, J. (2021). Requirements for a component-based modelling language for space missions. *VI SPANISH CONGRESS OF INFORMATICS (CEDI)*.