MIA, a reusable execution platform for space missions. A case study based on an Autonomous Flight Termination Unit.

Jesús Zurera [(1)], Alba Rozas [(1)], Miguel López[(1)]

[(1)] Sener Aerospace and Defense, jesus.zurera@aeroespacial.sener

ABSTRACT

As interest in space missions and the services they provide grows, the need to develop new platforms that can support the necessities of these services becomes more evident. High cost in time and resources are derived from Flight Software (FSW) development for each space mission. This effort, however, is hardly reusable because the software for each particular mission is tailored to its particular requirements. Thus, traditional FSW is tightly coupled, presents custom interfaces paired with data protocols, requires internal mission knowledge and presents interdependencies between components.

In this new space era, the price dictates the business. In order to increase the reusability, modularity and overall, ease the development of FSW, we have developed the sMart Integrated Avionics (MIA) platform. This platform significantly facilitates the development of space software and applications, such as the one presented as use case in this paper, an Autonomous Flight Termination Unit (AFTU). This autonomous launcher subsystem allows unmanned operations, increasing the maximum launch cadence, thus reducing the cost for space missions, facilitating that access to space is more easily and rapidly achieved by all types of agents.

1 INTRODUCTION

The MIA platform is presented as a versatile solution aiming to a variety of apace applications needs, ranging from the rigorous demands of space missions characterized by safety and functional criticality to the more flexible requisites of NewSpace applications, which prioritize rapid deployment and adaptability.

In this paper, we first present a general analysis of the MIA architecture, detailing the different parts it presents and the functionalities provided by each component. It offers a general description overview aligned with the platform's philosophy, emphasizing the modularity of its components and how they can be exchanged without affecting the rest, thanks to the isolation of its various software layers.

Next, a specific use case is presented involving an AFTU unit, demonstrating how a particular specification of the MIA platform was developed to meet the requirements of this system. The general characteristics of the chosen software are outlined, followed by a detailed explanation of how these software components facilitate the deployment of the capabilities needed for the development of an Autonomous Flight Termination System (AFTS).

Finally, two sections are dedicated to describing the implementation and execution specifics of the software and FPGA (Field-Programmable Gate Array) blocks for the specific AFTU system.

2 MIA PLATFORM

The MIA platform is composed by a set of interchangeable software components, FPGA blocks, and hardware devices in which it can be executed. MIA presents a highly decoupled layered architecture developed to operate and communicate using a set of standardized common interfaces between layers. This allows us to customize MIA by changing the components for each particular space mission implementation without affecting the rest, granting the needed modularity and reusability to the platform. This approach, allows the platform to be deployed over different physical devices such including Micro Controller Unit (MCU) or Multiprocessor Systems-on-Chip (MPSoC), providing the required modularity and flexibility to cope with different energy, memory or performance mission requirements.

The MIA execution platform is not a novel product being introduced within this paper; rather, its overarching description and analysis of its various components were initially presented at [1]. The philosophy underlying this platform remains consistent, to provide a FSW execution platform that excels at its flexibility and adaptability, from which novel outcomes can be derived. To aid in the comprehension of this paper, a review of the different components and their general characteristics is provided here. In subsequent sections, we will elaborate on how these distinct components have been tailored to serve specific purposes within a particular space application.

The principal layers that constitute the MIA architecture are shown in Figure 1 and are described below:

- The Hardware Layer includes the physical components of a System on Chip (SoC), including processor cores, FPGAs, memory devices, peripherals, and interconnect logic.

- The Temporal and Spatial Partitioning (TSP) Layer is responsible for providing temporal and spatial partitioning to the architecture. Space missions often entail specific applications of varying levels of criticality, such as Attitude and Orbit Control Systems (AOCS), Data Handling Systems (DHS), high-level applications, etc. Deploying a hypervisor enables us to leverage all physical resources, such as physical interfaces and memories, as well as the computational capacity of the processor cores, while maintaining isolation and criticality between different applications. This is obtained using software partition, which divide a computing system into multiple virtual environments. Each partition functions as an independent entity with its own operating system, applications, and allocated resources. This ensures that software partitions of varying criticality can be deployed on this layer without encountering interferences or interdependencies between them.

- The Operative System (OS) Layer provides the functionality to interface with the Hardware Layer either directly or virtualized by the hypervisor. It is also responsible for managing the priorities inside each Software Partition regulating the access to the different hardware resources.

- The Service Layer furnishes software services to applications. Within this layer, FSW solutions are deployed, providing a suite of standardized functionalities required by the majority of space applications. These services regard to application and time management, event listening and message exchange. Additionally, abstraction layers between different parts of the platform's software are included here, thereby isolating the development of the Real-Time Operating System (RTOS), common services provider, and the applications deployed over them. Special mention goes to Sener Services Layer Application programming interface

(SSLA), the abstraction layer through which application developers' interface to access the core services of the platform.

- The Application Layer is the domain where mission-specific functionality is implemented. Each of the various space applications that can be deployed to the platform requires multiple independent tasks utilizing the services provided by the Service Layer. It is within this context that these applications are developed to meet the objectives of different space missions. Examples include command ingestion and telemetry transmission, execution schedulers, mode managers, or user developed applications for managing external subsystems such as batteries, solar panels, or sensors and actuators. The objective of this platform would be to establish a library of applications previously developed by users, each with specific functionalities as described earlier, tailored for particular subsystems of a satellite, for instance. This approach enables us to offer future users ready-made applications, pre-developed and ready for integration into the platform. Consequently, the platform, along with the application, would evolve into a potential ready-to-use configurable product.

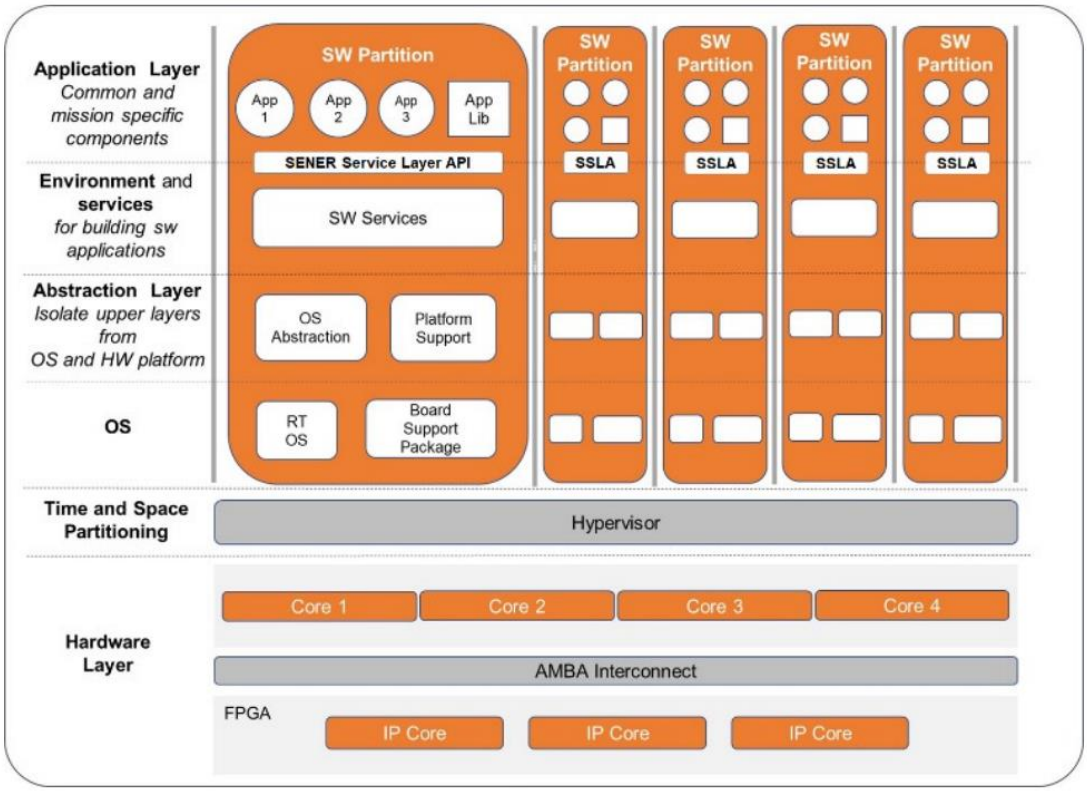

Figure 1 MIA generic architecture

SSLA is central to the design of MIA, it comes along with its own Software Development Kit (SDK), and it presents developers with a set of common services and functionalities, useful in the context of space missions, for deploying their own mission FSW. These services and functionalities are tightly coupled with the presence of core Flight System (cFS) [2] in the platform, with SSLA acting as a façade for the most useful of its Application Programming Interfaces (APIs). SSLA also provides some own services and a standard execution flow for applications. This SDK enables developers to create and deploy applications with ease. Firstly, it enables the assignment of processes priorities, allowing for the optimization of task execution based on predefined importance. Secondly, it facilitates the establishment of clear interfaces between tasks, following a publication subscription communication methodology. Lastly, it provides a rich repository of function libraries both providing

a fixed set of services that most FSW requires as well as allowing user-specific ones, enhancing the tasks capabilities without increasing its complexity. The services provided by the SSLA library comprise the following functionalities:

- Event Handling Services: Handling various types of events is the focus of this service. It provides methods for sending debug, informational, error, and critical events, each tailored to specific event types, enabling effective event management and debugging.

- Logging Services: Logging functionality is consolidated within this service, allowing messages to be logged to the system log for record-keeping and diagnostic purposes.

- Application Management Services: Designed to manage applications within the system, this class facilitates operations such as retrieving application IDs and names, registering child tasks, and managing task-related operations within the system architecture.

- System Control Services: Responsible for system-level control, this service offers functions for initiating system resets, providing a mechanism for system-wide management and control.

- Message Handling Services: Central to message handling operations, this service provides a comprehensive suite of functions for message subscription, retrieval of message IDs and command codes, registration of handlers for different types of messages, message reception, transmission, and time retrieval.

These services collectively form the backbone of the SSLA library, providing developers with a powerful set of tools for building and managing complex space applications efficiently.

3 AFTU ARCHITECTURE DESCRIPTION

Multiple sets of space mission applications can be deployed using different MIA platform configurations. For the purposes of this paper, we have selected a particular case study based on an AFTU. A set of power independent, sensor isolated AFTUs conform an AFTS, a system that checks along the flight the correctness of a launcher trajectory, outputting a unique and coherent terminate signal. This autonomous launcher subsystem allows unmanned operations, increasing the maximum launch cadence, thus reducing the cost for space missions. This system has been developed under the scope of the SAFEST project, which stands for Smart Avionics for Flight Termination Systems. This project has been undertaken by a consortium of European companies, with leadership provided by SENER Defense and Space. The various contributions from the consortium members that have facilitated the development of the MIA platform will be presented alongside their respective contributions.

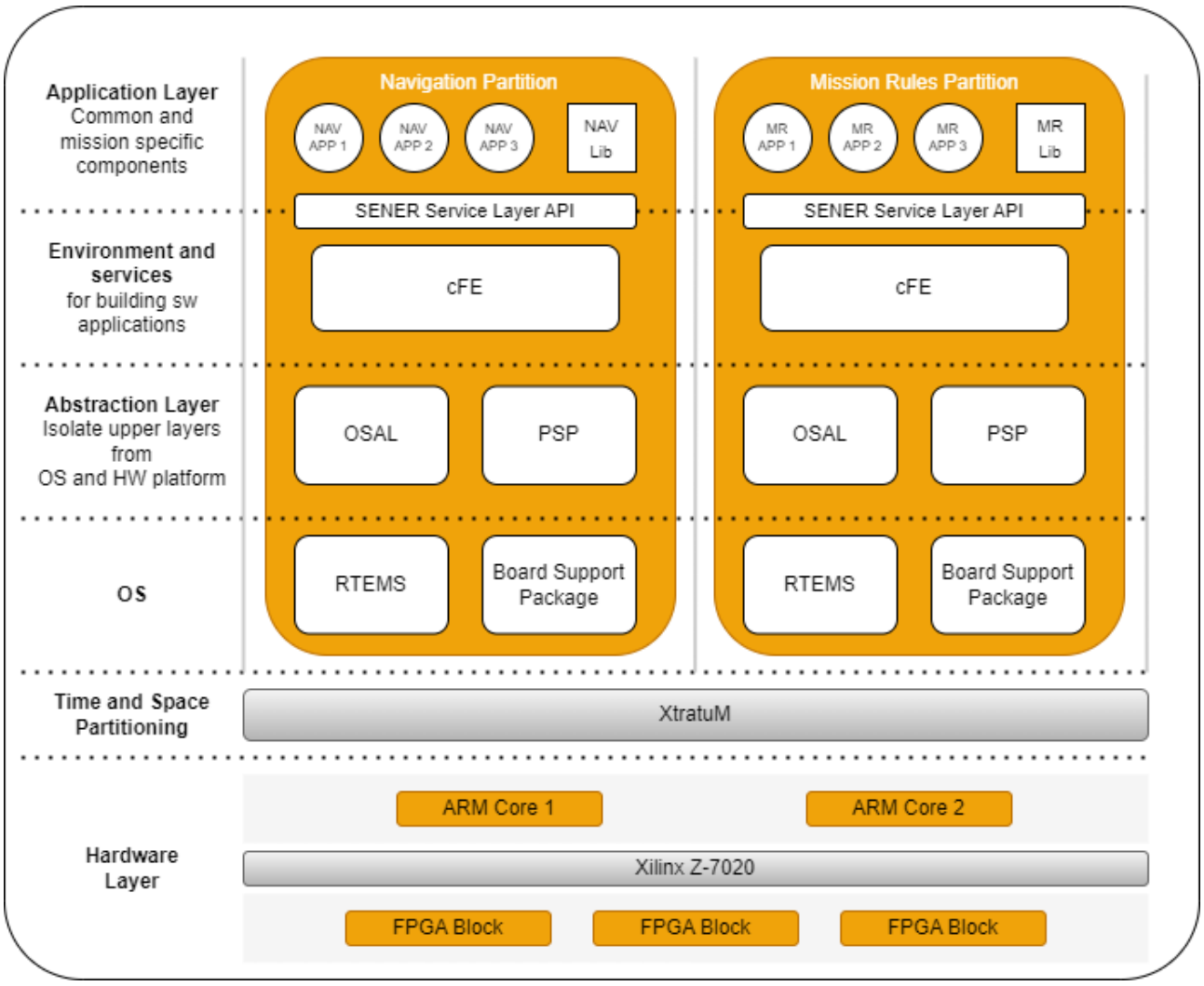

Figure 2 MIA specification for the AFTU

For this particular use case, we have developed a MIA specification that follows the architecture shown in Figure 2. It is developed to be deployed over the Xilinx MPSoC Z-7020. At software level, the structure designed for MIA to fulfill the needs of an AFTU and the role that its different parts carry out is the following.

3.1 XtratuM Hypervisor

FentISS XtratuM Hypervisor for Time and Space Partitioning [2][3]. The hypervisor plays a pivotal role in enabling the deployment of the two primary applications of the AFTU: the Navigation Partition and the Mission Rules partition, which are presented in more detail in section 4. These two FSW Applications could be deployed on separate On-Board Computers (OBCs). However, through the utilization of the Hypervisor to deploy them as software partitions within the same hardware, the full potential of the MPSoC is utilized. By consolidating these applications onto a single hardware platform, various benefits are realized. Firstly, communication between the applications is streamlined, as they can directly interact with each other within the confines of the MPSoC, making use of inter-partition communication. This eliminates the need for complex inter-OBC communication protocols, reducing overhead and potential points of failure. Additionally, resource utilization is optimized, as the shared hardware resources of the MPSoC are efficiently distributed among the deployed partitions. This ensures that computational resources are fully utilized, enhancing overall system performance and efficiency. Moreover, deploying both applications within the same hardware environment simplifies system debug and testing, as the system is a centralized in a unique platform.

3.2 RTEMS RTOS

RTEMS (Real-Time Executive for Multiprocessor Systems) as Real Time Operating System (RTOS). RTEMS is deployed over the XNG hypervisor, and the synergy between these two software layers has been proven across numerous projects both, making use of the capabilities of Symmetric Multiprocessing (SMP) or single-core configurations, such as that of this platform. In the MIA platform the applications are managed by the OS within each partition. RTEMS makes use of its scheduler module to handle the execution flow of the different tasks. The concept of scheduling in real-time systems dictates the ability to provide immediate response to specific external events, particularly the necessity of scheduling tasks to run within a specified time limit without surpassing it. The scheduler's sole purpose is to allocate the resource of processor time to the various tasks requiring execution time at the same software partition.

By utilizing this scheduler, we can ensure the continuous execution of the highest priority task pending execution at any given time. This functionality aligns with a configuration option within RTEMS known as preemption. In our particular use case, there are very high temporal constraints due to rapid data aging. For instance, consider the transmission of a navigation solution from the Navigation partition to the Mission Rules partition. From the moment data is received from the sensors until the AFTU produces a correct termination signal, the maximum elapsed time must be less than 10 ms. Such a temporal requirement requires that the execution of tasks responsible for initial trajectory calculations and outputting to the Mission Rules partition must have the highest priority. This ensures their capability to preempt lower-priority tasks, thereby meeting the time constraints effectively.

3.3 cFE

cFE the core of NASA's cFS for the FSW framework. The cFE provides a set of core services including Software Bus (messaging), Time, Event (Alerts), Executive (startup and runtime), and Table services. The cFE defines an API for each service which serves as the basis for application development. Both, cFE and RTEMS are particularly tailored to fulfill the necessities and certification requirements that space missions present. For this project, a space profile cFE/RTEMS is developed. The reduction of features such as file systems, embedded shells, or dynamic loading, make possible to create a Qualification Data Package (QDP) tailored for applications categorized under ECSS software criticality C. This QDP is specifically designed for deployment on the Xilinx Z-7020 platform by the consortium company embedded brains.

3.4 SSLA

SSLA operates as a service provider to the Application Layer. SSLA acts is an abstraction and isolation layer, providing a structured and standardized interface for applications to access the platform's services and functionalities. Applications are separated from the specific platform implementation details such as cFE, Programmable Logic (PL), and communication interfaces. Moreover, any modifications made to the lower layers of the platform have minimal impact on the applications. Furthermore, employing SSLA allows developers to view the remaining components of the MIA platform as a black box, simplifying the development process and reducing complexity.

4 AFTU SOFTWARE PARTITIONS

As presented in the previous section, to fulfill the requirements of the AFTU, two Mission Partitions are developed. A Mission Partition is a set of applications intended to work together as a whole, including common variables well known for each application as well as a set of priorities defined to allow critical task behavior to work correctly. For the AFTU, the two Mission Partitions are the 'Navigation' and the 'Mission Rules', each of them executed in a separate hypervisor partition.

The Navigation partition encompasses the system's guidance functionality, incorporating sensor data ingestion to produce a precise navigation solution. This partition is responsible for processing real-time sensor inputs and executing algorithms to ensure precise and accurate position location throughout the flight trajectory. In parallel, the Mission Rules partition operates as a flight correctness and trajectory verification system. It continuously monitors the flight parameters against predefined mission rules and criteria, verifying the adherence to desired flight paths and safety protocols. These parameters not only come from the inner Navigation partition, but also from others tracking inputs that the user can send to the system. By segregating these functionalities into distinct partitions, the hypervisor ensures the isolation and independent operation of the Navigation and Mission Rules systems. Making use of the flexibility of the platform, the AFTU is also designed to be modular in its components, allowing the user to change its components, such as the sensors and Navigation partition, as long as the navigation parameters that it sends are kept the same. The opposite is also true, the system could be modified by the user to add its own Mission Rules partition, maintaining the Navigation partition, or relaying on external tracking inputs.

4.1 Navigation Partition

The Navigation partition produces an estimation of the positioning state vector and an indicator of the quality of the estimation. To do so the navigation application processes and integrates the information coming from two subsystems: an Inertial Measurement Unit (IMU) and Global Navigation Satellite System (GNSS). The data fusion algorithm implemented is a Loose Extended Kalman Filter (EKF) [5][6]. The architecture and high-level data flow of the navigation is described in Figure 3.

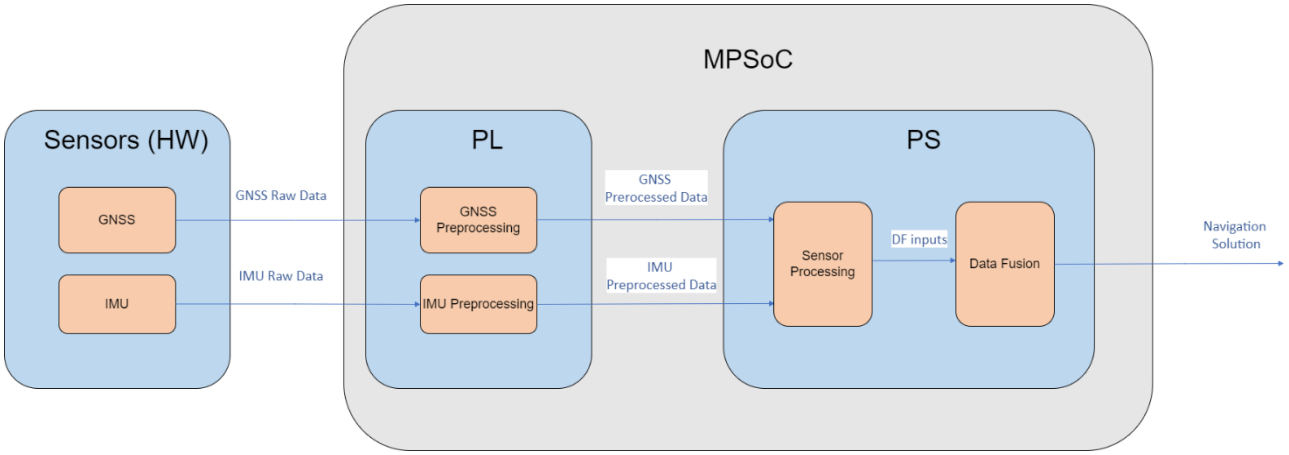

Figure 3 High-level Navigation Partition data flow

To deploy this architecture on the MIA platform, a set of SSLA applications and FPGA blocks have been developed. The description of the FPGA blocks will be addressed in a subsequent section. Here, our focus is to showcase the SSLA application architecture that has been developed to replicate the high-level behavior required to meet the functional requirements of this partition. This architecture is shown in 4. It is important to note that even though in the figure two FPGA blocks appear, this is just for clarity in following the execution flow of applications, they refer to the same physical device. It is also important to notice from the figure the presence of three different types of communication. The first, in red, the hardware interruption that triggers a cyclic execution, the second, in green, the command message, which carries information and triggers applications execution, the third, in blue, the data messages, which only store information inside an application pipe to be read when awaken by a command message. Lastly, the thin black arrows represent the FPGA related communication. This communication is described in more detail in section 5.

The execution of these applications constitutes a cycle of operation. This execution cycle is triggered by the arrival of a tick from the FPGA, which, through a hardware interrupt, initiates the execution of an application. This interruption is linked to the arrival of a new data packet from the sensors, in this case, from the IMU, which has the highest frequency. In this implementation, the IMU message reception is set to 50 Hz. The arrival of a new GNSS is asynchronous to the system, and, as it has a lower frequency and priority than the one from the IMU, it does not trigger any hardware interruption and it is read in the following cycle. The arrival of the interruption initiates an internal communication cycle using the internal communication bus, where this set of sensor data is distributed among the different applications that require them. Within this cycle, a navigation solution is obtained and sent through the FPGA to the Mission Rules partition.

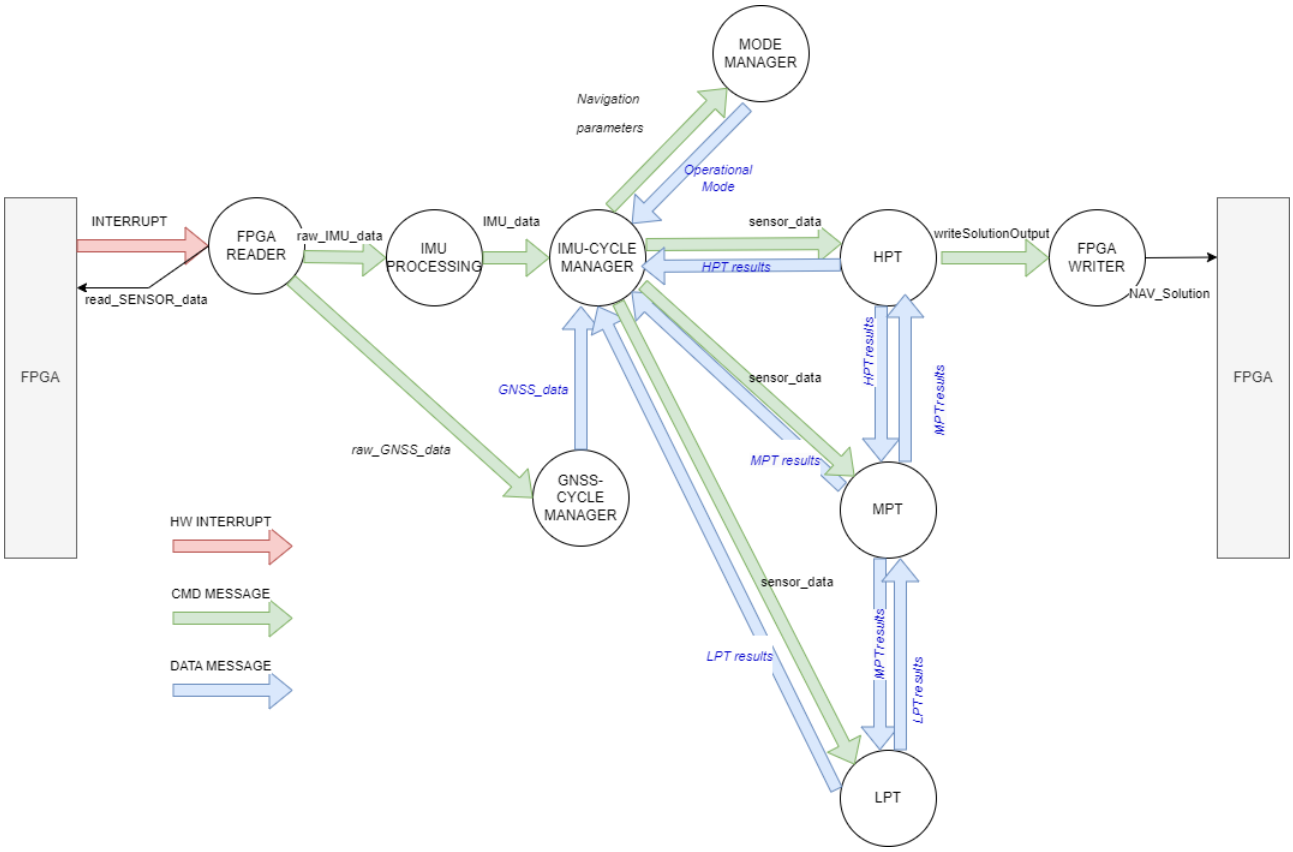

Figure 4 Navigation Partition SSLA application architecture

Three main SSLA application types are developed. FPGA communication, scheduler management and navigation algorithm function applications.

FPGA communication applications: These applications are specifically designed to manage communication with the FPGA blocks within the system. They facilitate the exchange of data between software applications and FPGA logic, ensuring seamless integration and coordination between the two components. They are designed to be highly reusable and decouple the rest of the SW from the external HW devices intricacies, being the only applications that would require modification in case of changing the sensor or actuator selection.

- FPGA Reader: This application receives a HW interruption triggering the system cycle. It reads sensor data from the memory, which has been previously written by the FPGA through the usage of Direct Memory Access (DMA). Then, the application sends messages making

use of the SSLA communication services to the corresponding applications, as illustrated in Figure 4. In case of malfunctioning at retrieving the data, the application has configured a set of informational and error events that can be raised at any time.

- FPGA Writer: This application sole purpose is to receive the information that shall be outputted each cycle either to the Mission Rules partition or as telemetry using one of the configured external interfaces.

Scheduler Management applications: These applications interface between the raw and processed data and the navigation algorithm functions. They control the operational modes and application communication sequences.

- Mode Manager: The Mode Manager application implements a state machine that receives a set of parameters as inputs and decides on the system's mode transitions. These parameters are related to navigation, such as the validity of sensors or the results of navigation algorithm functions.

- IMU Cycle Manager: The IMU Cycle Manager application is responsible for orchestrating the execution sequence of algorithm applications with navigation functions and the Mode Manager.

- GNSS Cycle Manager: The GNSS Cycle Manager application is triggered to execute upon receiving a GNSS message. Its mission is to propagate the data received from this sensor to the IMU Cycle Manager, which will encapsulate it as a new message in its communications with the navigation algorithm applications.

Navigation algorithm applications: These applications encompass all operations performed on sensor data to derive a navigation solution. These functions are autogenerated from graphical programming environment and embedded within various applications listed below. These applications are where the majority of CPU execution time per cycle is allocated, thus they are designed in such a way that the execution of lower-priority functions will never halt the execution of a function with stricter temporal requirements. Priorities are set regarding which sensor data the application computes. The higher priority applications work with the IMU, which data aging, the time that it takes for a data to be obsolete, is faster in comparison with GNSS due to their different reception frequencies (50Hz for the IMU, 2Hz for the GNSS).

- High Priority Task (HPT): First level priority. Receives a message from IMU Cycle Manager with sensor data and previous results from the rest of navigation functions. It executes its algorithm function and propagate its results. Due to time constrains and strict limitation on data aging in an AFTS, HPT is the application which commands the execution of FPGA Writer, ensuring the lowest navigation solution transmission time.

- Medium Priority Task (MPT): Second level priority. Receives data from IMU Cycle Manager with sensor data and HPT results, computes it, and propagates its results.

- Low Priority Task (LPT). Third level priority. Receives data from IMU Cycle Manager with sensor data, HPT and MPT results, computes it, and propagates its results.

4.2 Mission Rules Partition

The Mission Rules partition process the tracking input information, update the state accordingly and detect if any mission rule has been violated. The mission rules can process up to three tracking inputs;

the first one would come from the Navigation partition and the other two are optional, could be provided by other AFTU or by the user. This architecture is presented in Figure 5.

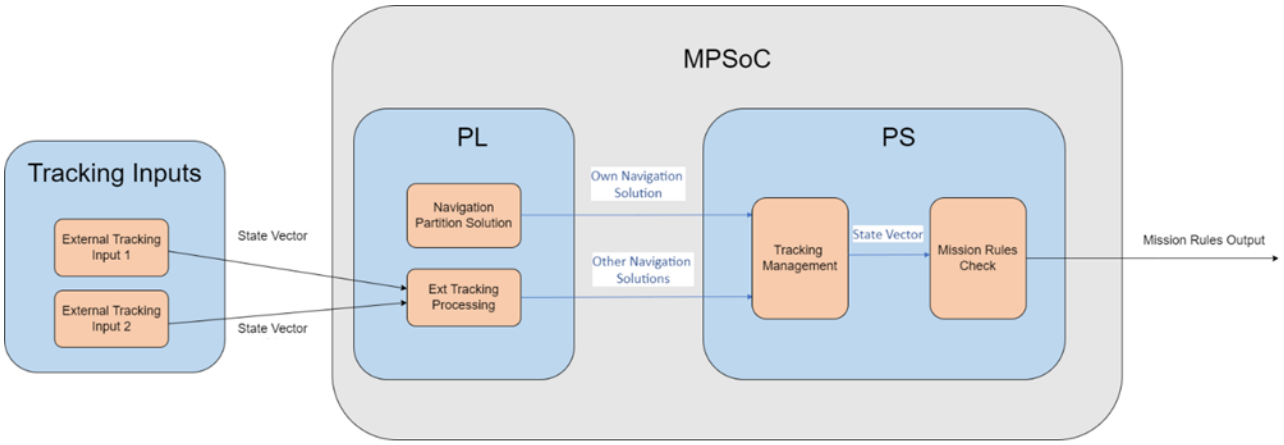

Figure 5 High-level Mission Rules Partition data flow

Integrating this logical architecture into a set of SSLA applications and FPGA blocks on the MIA platform is straightforward. From the perspective of the software architecture to be developed, it is resolved with four applications. The two FPGA read and write applications, which are reusable from the previous partition and reconfigured to receive and write the desired data. A Mode Manager application, which receives parameters and computes the system's operating mode. Lastly, the Mission Rules Checker application, which checks the correctness of the trajectory rule by rule. As well as the Navigation partition, this partition also relays on a hardware interruption coming from the FPGA to trigger its execution cycle. This hardware interruption is paired with the Navigation partition writing the navigation solution to the FPGA. In this case, this communication is set to a frequency of 50Hz. This application configuration is shown in Figure 6.

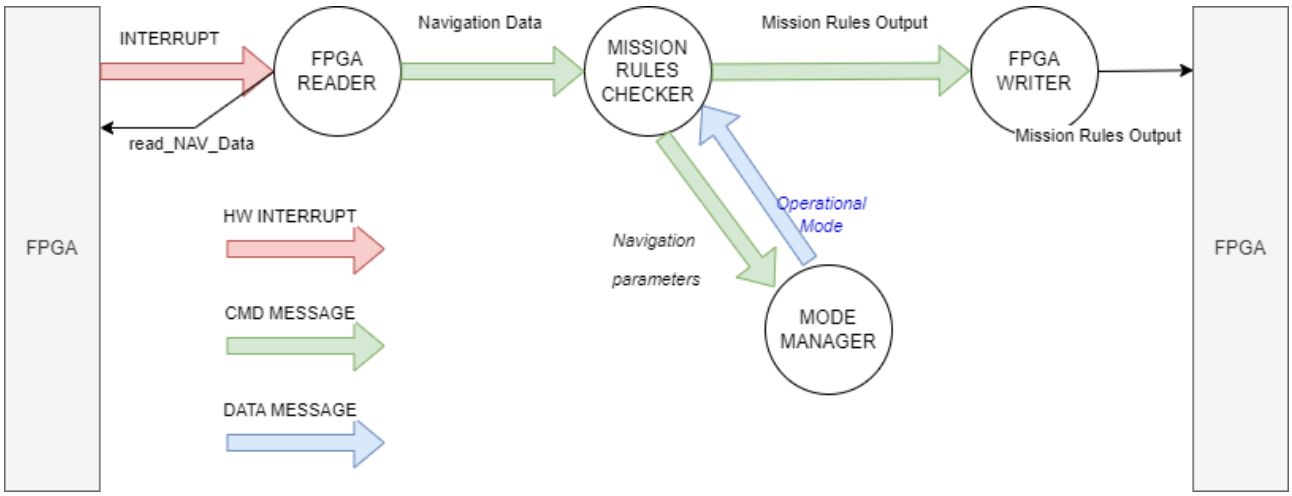

Figure 6 Mission Rules Partition SSLA application architecture

This Mission Rules partition is designed and implemented aiming to achieve a lightweight and user-friendly application architecture for an Autonomous Flight Termination System (AFTS). This design choice is deliberate. The goal is to empower the user of an AFTS system with configuration capability over the Mission Rules partition. From the perspective of an AFTS product, this approach makes sense, as there are numerous parameters that are dependent on the launcher on which the AFTU is installed, necessitating user configurability.

In this regard, two types of possible configurations have been considered, ranging from lower to higher levels of system modification:

- Modification of specific parameters related to launch characteristics such as number and definition of tracking inputs, launcher characteristics or trajectory definition.
- Integration of a new Mission Rules SW partition, based on the one provided in this project but modified, compiled, and introduced into the system to be orchestrated by the hypervisor.

5 AFTU FPGA BLOCKS

Since, in this use case, it is designed to be executed on a Xilinx MPSoC Z-7020, the MIA platform has built-in features for allocating different functionalities in the FPGA (Programmable Logic, PL) and regular software in the ARM cores (Processing System, PS). In particular, the SSLA API and service layer encapsulates all accesses to the FPGA registers and modules in a clear boundary (SSLA Lib).

The information exchange between the PL and the PS in the MIA architecture is performed using Direct Memory Access (DMA), this interaction is presented at Figure 7. This engine allows to transfer data directly from the FPGA to the MPSoC memory. In particular, it allows sending data from the PL to the Double Data Rate (DDR) Random Access Memory (RAM) without requiring the PS. The MIA platform assumes that the FPGA logic and circuitry is designed in a structure of several register blocks, accessible through the Advanced eXtensible Interface (AXI) bus, which configures and controls a set of different FPGA subsystem. The DMA, its control registers, status registers, buffer length (size of the transferences), destination address, etc, the different sensors and actuators connected through external interfaces and the management of Telemetry and Telecommand (TMTC) are configured using the AXI bus.

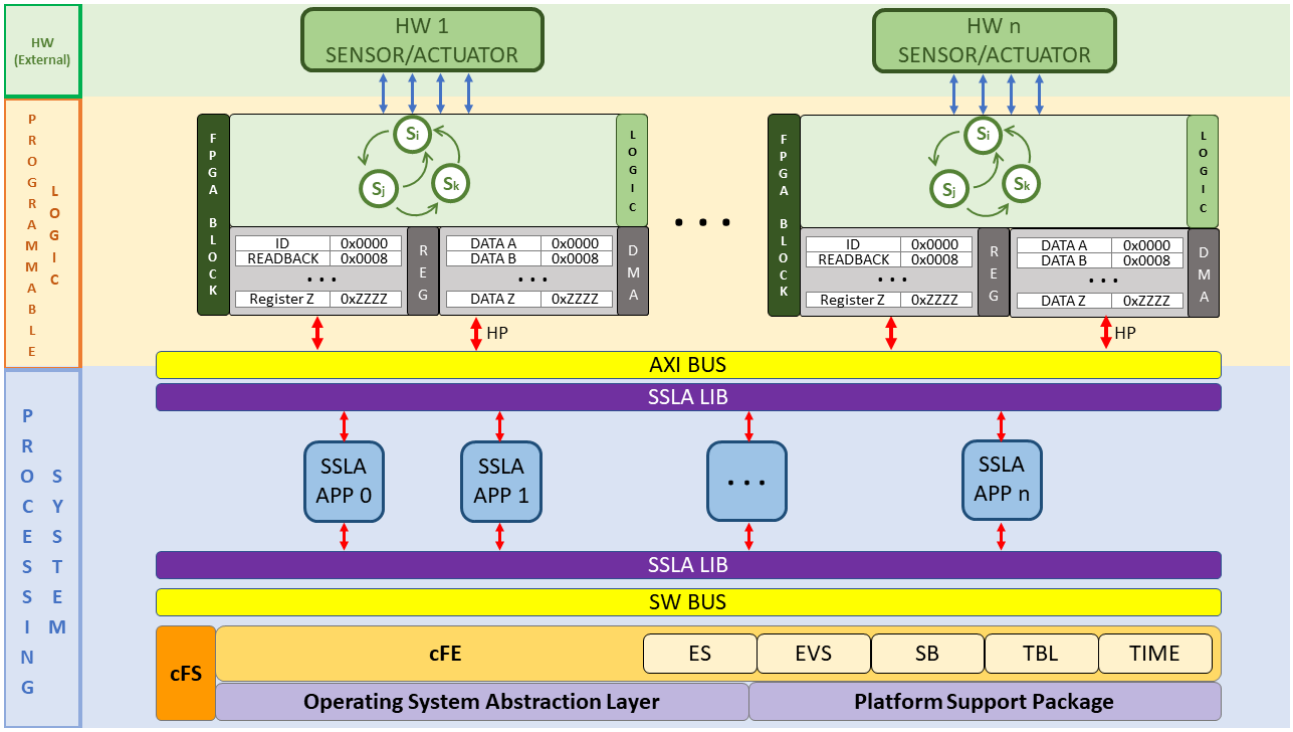

Figure 7 PL/PS Interaction

Although this is mission-dependent, the common approach both in MIA and in most FPGA-equipped embedded systems is to allocate the low-level functionality of interfacing with external sensors,

actuators and subsystems in the FPGA. This reduces the software processing load and allows the applications to be decoupled from the particular timing constraints of the different external hardware units.

In this use case, the Navigation partition software requires the ingestion of the data produced by two sensors, as described in 4.1, a IMU and a GNSS. These sensors are connected to the PL through a serial interface RS422. Two FPGA blocks are developed to receive these data (implementing the necessary serial protocol, flow control, etc.) and perform a pre-processing or conditioning on the data. Then, the DMA engine is used to store the data retrieved and pre-processed from the sensors in the application developed to read the data from the FPGA. With this approach, the software application does not need to deal with the specifics of the serial protocol and timing, nor receive unnecessary data that it won't need for its functionality. The app would just read its required data from the memory on its own decided schedule, making the app lighter and the guaranteeing of the global software determinism much more feasible. In addition, this approach allows to change the particular sensors used, both IMU ad GNSS, only modifying the FPGA block. The modification would only require adapting the new received data to the frequency and generating the same message output format to the PS, leaving the application software without changes.

As previously mentioned in section 4, both software partitions require a hardware interruption to start their execution cycle, delegating their periodicity to an external input. In this particular design, the Navigation partition interruption is dependent on the IMU sending messages through the RS422. These messages are produced at 200 Hz by the sensor used. When the FPGA has received four IMU packages, it processes them and writes the data to the PS memory at 50 Hz.

When using the hypervisor, the FPGA is treated as another hardware resource from the point of view of each partition. A special FPGA block is used to exchange data between partitions in a secure way, allowing partitions to share data without directly interfacing with each other. In the case of the Mission Rules partition, the interruption that initiates the execution cycle depends on the Navigation partition writing the navigation solution to the FPGA.

6 MIA SUPPORT FOR HIGH-LEVEL ALGORITHM INTEGRATION

Throughout this work, the motivations for using a platform like MIA for rapid, effective, and secure deployment of FSW have been developed. Here, is shown how using this platform reduces development time and cost.

As presented in previous sections, the MIA platform is designed in a versatile way so different space applications can be deployed over it. As of today, one of the most commonly required space application are guidance, navigation, and control (GNC) systems. These systems present complex functionalities, tightly coupled to the spacecraft characteristics in which they are deployed. Traditionally, this generates mission specific software that is hardly reusable. As of today, the development of these algorithm systems is ever more dependent on simulator environments such as Simulink or high-level algorithmic languages such as phyton or MATLAB. They are used for modeling, simulating, and analyzing dynamic systems with complex algorithm functions. These tools can provide a visual, block-based interface where users can build models by connecting predefined blocks representing different system components. From these block-based diagrams or algorithm descriptions, a non-experienced software developer user can run automatic code generation tools, which ultimately generate a set of files in the programming language chosen by the user. These files consist of a series of function declarations and structures that, when properly initialized with the

appropriate values and executed, replicate in software the behavior previously designed in the graphical system.

Starting from this set of files containing source code and declarations of the algorithmic functions produced by each block, a direct integration of these functions and interfaces can be achieved within the MIA platform. Each of the designed blocks can generate source files ready to be added to different applications with the granularity desired by the user. The simplest integration involves embedding all functionalities executed at the same frequency into a single block, from which a function is automatically generated,. The inputs and outputs of the function can be used as the definition of the messages to which the SSLA application must subscribe and publish, respectively. This integration is presented in Figure 8.

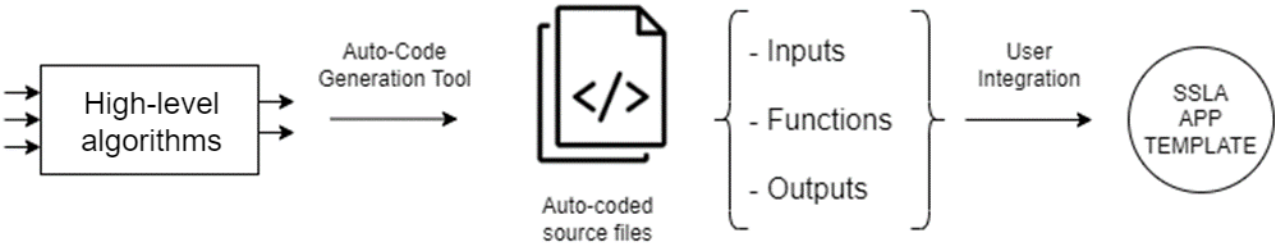

Figure 8 High-level algorithm blocks to SSLA application integration

As presented in the previous section, both the AFTU and the platform itself are designed to be modified either by configuring specific parameters to define the operation of a launcher or by replacing the Mission Rules partition. In the latter case, users are provided with a template of the source code for this partition.

Given the pre-developed and provided sensors, actuators, FPGA blocks, and SSLA applications that interact with them to obtain and deliver data externally, the MIA platform is designed so an inexperienced user can develop communications and applications to replicate the behavior of various algorithmic functions on this data. Following the example of the software partition described in Figure 6, a set of templates and tutorials are provided to facilitate the integration of this auto-generated software. The Mission Rules Checker application, along with its corresponding algorithm library, is provided to the developer in the form of a template. Here, the developer can integrate the functions generated by their auto-generation tool. The user only needs to define the inputs of their module, equivalent to the input interfaces defined in their system, and call the auto-coded function embedding the functionality of the system. The execution of this function will return a set of values that will be parsed into the Mission Rules Output message, thus completing the process of integrating auto-generated software with the MIA platform.

Once all the software is integrated, the tools of the MIA platform include all the necessary configuration files to compile the entire system and generate an executable in the form of a software partition. This partition can then be integrated into the platform and loaded by the hypervisor as a new software partition. This process is displayed in Figure 9.

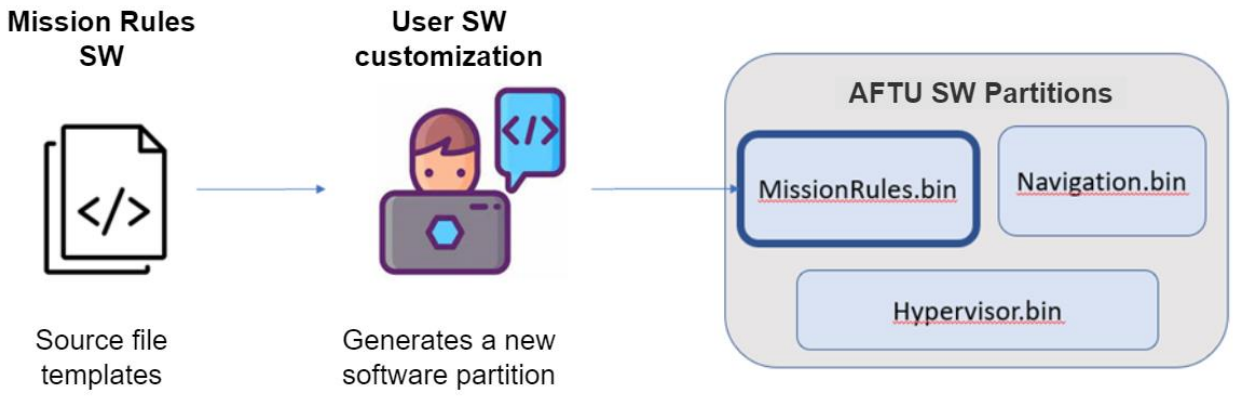

Figure 9 User Software Partition customization

7 CONCLUSIONS

Portrayed in the AFTU use case, the advantages of using the MIA platform as an avionics solution for a real scenario are proven. Deploying two distinct software partitions, Navigation and Mission Rules, ensures their isolation, enabling easy modification or replacement, thus enhancing the platform's modularity and flexibility. The different SSLA applications with communication functionality with the FPGA, as well as the FPGA blocks themselves, can be reused in future projects for interfacing with these same sensors, or slightly modified for other sensors of the same nature. The templates and tutorials for integrating autogenerated functions into SSLA applications, with a clear approach of inputs as reception messages, autocoded functions as main execution and outputs as publication messages, reducing the software development complexity.

In summary, this paper delves into the advantages that using the MIA platform provides to developers and companies in the space and avionics sector. The ability to deploy different configurations of space software with different criticalities over one platform, the incremental added value that new reusable applications provide, and the platform's modularity make MIA an advanced, easy to use, low-cost and modular avionics solution.

8 ACKNOWLEDGMENTS

This work is carried out in the frame of the SAFEST Project with Sener as the coordinator of a joint effort of many individuals and organizations. This project has received funding from the European Union's Horizon Europe research and innovation programmed under grant agreement No 101082662.

We greatly thank of the great work of all the consortium members, in particular in particular to fentISS (hypervisor) and embedded brains (operating system), the European Commision officers and reviews and the business development department from Sener, which have believed in and firmly committed to the project.

9 REFERENCES

[1] Santiago Lozano, Ana Rodríguez, Carlos Rodríguez, and Miguel López, *MIA: European Multi-Purpose Space Platform using cFS and TSP*, Flight Software Workshop, 2022.

[2] McComas, D. (2012, November). NASA/GSFC's Flight Software Core Flight System. In Flight Software WorkshopFlight Workshop (No. GSFC. CPR. 7525.2013).

[3] Masmano, M., Coronel, J., Balbastre, P., Crespo, A., Simó, J., & Peiró, S. XtratuM hypervisor for mixed-criticality systems.

[4] Masmano, M., Ripoll, I., Crespo, A., Metge, J.J., and Arberet, P. Xtratum: An open source hypervisor for TSP embed- ded systems in aerospace. In DASIA 2009. DAta Systems In Aerospace., Istanbul (Turkey), May 2009.

[5] S. Ramirez, S. Diaz, C. Fernandez, NAVIGA: Multi-Purpose European Space Navigation Unit, IAC 2022

[6] M. Sanchez, S. Ramirez, C. Tato, S. Caporossi, Next Generation Autonomous Flight Termination System (aFTS) for Launchers, IAF Space Transportation Solutions and Innovations Symposium, 2022.