Verification and Validation of Dynamic Reconfiguration in Integrated Modular Avionics with a MATLAB/Simulink ARINC 653 Blockset

Bojan Lukić
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
Email: bojan.lukic@dlr.de
0009-0002-4286-1901

Umut Durak
Institute of Flight Systems
German Aerospace Center (DLR)
Braunschweig, Germany
Email: umut.durak@dlr.de
0000-0002-2928-1710

Matthias Klimmek
Software Engineering
SYSGO GmbH
Klein-Winternheim, Germany
Email: matthias.klimmek@sysgo.com

Neelakanta Erabhovi
Systems Engineering Division
National Aerospace Laboratories
Bengaluru, India
Email: neelakanta@nal.res.in

Umayr Jahagirdar
Systems Engineering Division
National Aerospace Laboratories
Bengaluru, India
Email: umayr.jay@outlook.com

Manju Nanda
Systems Engineering Division
National Aerospace Laboratories
Bengaluru, India
Email: manjun@nal.res.in

***Abstract*—The development of safety-critical avionics systems demands a great rigor for structured engineering practices. Model-Based Design (MBD) has emerged as a methodology, where models with their formal/semi-formal basis are not only used to specify and eventually automate the implementation of the systems but also provide means to produce evidence regarding their dependability. MBD fosters the tight coupling between system design and simulation. Simulation, the execution of the models, enables virtual verification and validation (V&V). Thereby the feedback cycles are shortened and faster design iterations are enabled. In this paper, we present a virtual V&V approach for the dynamic reconfiguration features in an Integrated Modular Avionics (IMA) system using MBD. The IMA platform and the applications are modeled using the MATLAB/Simulink ARINC 653 Blockset. The model simulation is used for the V&V of the reconfiguration mechanisms that are designed for fault adaptation. The demonstration consists of a model of the IMA platform and a simulation for the reconfiguration of platform components during runtime. The fidelity of the simulation towards its intended hardware environment is increased with parameter considerations from an exemplary embedded system. We report agile development using MBD and V&V in simulation with results matching tests from the referenced embedded hardware.**

***Index Terms*—Integrated Modular Avionics (IMA), Rapid Prototyping, Dynamic Reconfiguration, ARINC 653, Simulation, MATLAB/Simulink**

I. Introduction

The increasing complexity and safety requirements of modern avionics systems demand dependable system engineering practices that provide evidence for system validation and certification. Traditional Hardware-in-the-Loop (HiL) testing methods, while highly representative, are often time-consuming and costly, posing challenges for short development cycles [1]. To address these challenges, systems engineering along with simulation has emerged within Model-Based Development (MBD), enabling engineers to verify and validate systems virtually before deploying on physical hardware [2].

This paper discusses a class of safety-critical avionics systems, namely Integrated Modular Avionics (IMA). With standards such as ARINC 653, IMA provides a flexible framework for partitioning and hosting different avionics functions on the same hardware. With the advent of such flexible avionics frameworks, considerations for more elaborate system operations arise. One of the more prominent capabilities of IMA systems that are considered for future adoption is dynamic reconfiguration [3]. In the context of IMA, dynamic reconfiguration refers to the avionics system adapting to changing mission needs or failures, enhancing its reliability and safety. However, ensuring the correctness and robustness of such elaborate functionalities for a safety-critical system like IMA requires meticulous testing and validation. To shorten the feedback cycles in design iterations of the systems, virtual testing methods can be used for verification and validation (V&V). The use of prototypes in simulation software, such as MATLAB/Simulink, offers a promising way forward. This practice facilitates early validation and certification preparation without the immediate need for HiL testing.

This paper presents a methodology for V&V of reconfigurable avionics systems by simulating early prototypes. The IMA platform and the applications running on it are modeled in MATLAB/Simulink using an ARINC 653 Blockset. For demonstration, a *PowerPC VPX3-152 T2080* board running PikeOS as the ARINC 653 operating system (OS) is used as reference architecture for the simulation. Provided that the developed models are sufficiently accurate, we show that V&V for dynamic reconfiguration can be achieved with high fidelity.

The remaining work is structured as follows. In section II, we introduce the definitions, technologies, and standards concerning simulation engineering for dynamic reconfiguration in IMA. In section III, we present current research for the development and simulation of dynamically reconfigurable avionics systems. Section IV demonstrates a simulation environment for dynamically reconfigurable avionics system. The results are discussed in section V before concluding the work in section VI.

II. Background

The background spans three subsections. The first presents standards and guidance documents with technical considerations for the V&V of safety-critical avionics systems. The second subsection presents the software and tools used for the simulation environment demonstrated in this paper before defining terms and technologies regarding simulation and dynamic reconfiguration in IMA in the last subsection.

A. Standards and Guidance Documents

The standards and guidances presented here are relevant either for the technical implementation of the desired system or simulation practices for V&V activities. These can eventually be considered for certification. Following is a selection of applicable references.

ARINC 653

ARINC 653 is a standard that proposes an APplication/EXecutive (APEX) for managing the execution of software in IMA platforms [4]. In particular, it introduces the use of time- and space-partitioning for applications running on the same hardware in a real-time operating system (RTOS). ARINC 653 specifies how applications in a system can be mapped to isolated partitions, which ensures that each partition operates independently and securely, preventing interference with other partitions. These partitions can be statically or dynamically configured, and the standard defines a scheduling mechanism that ensures deterministic behavior for real-time applications [4]. The avionics system which has been set up for the use case of the paper at hand adheres to ARINC 653. More detailed information about the role of ARINC 653 for dynamic reconfiguration is discussed in a later subsection.

DO-297/ED-124

The DO-297/ED-124 serves as a guideline for the processes, tasks, and deliverables to be adhered to throughout the development lifecycle of IMA systems [5]. It addresses a range of areas such as requirements engineering, software design, V&V, configuration management, and system integration.

DO-297/ED-124 discusses reconfiguration functionalities of IMA systems in section 3.7.1.1, stating that "[c]onfiguration data [...] is used by the IMA system to: [...] **activate** or **deactivate** modules, resources, or functions, e.g., adaptation of the software to several aircraft configurations using option-selectable software or data. [...] [C]onfiguration data may be used [for] adaptation of the software to several aircraft configurations using option-selectable software [...]" [5].

DO-297/ED-124 therefore supports reconfiguration of components in avionics. That is, the activation or deactivation of modules or applications specified in a configuration. Moreover, though not explicitly formulated, the guidance does not rule out reconfiguration during runtime. The RTOS used for the demonstrated prototype in this paper, PikeOS, embodies the concepts that are introduced in the DO-297/ED-124.

DO-331/ED-218

The DO-331/ED-218, also known as the MBD and Verification Supplement to DO-178C, provides additional objectives and guidance for using MBD in projects that adhere to the DO-178C and DO-278A. As per DO-331, the model simulation environment encompasses the model simulator along with the necessary supporting tools and operational framework to carry out specific verification tasks. Section MB B.8 states that models are abstract representations of particular system aspects and, therefore, are not classified as tools that require qualification. However, model simulators might need tool qualification depending on the system requirements [6].

We demonstrate how executable Simulink models, coupled with RTOS configurations, can be verified against high-level requirements, as promoted in DO-331. The use of HiL simulations ensures bidirectional traceability and test coverage.

ARP4754

Prototypes are defined in ARP4754A as the models of the desired system that enable user interaction to uncover missing requirements, unwanted behavior, and other potential problems. ARP4754A outlines various techniques to support validation including tests, analysis, and traceability. Simulation is also proposed as a testing approach. The standard states, though, that "care should be exercised to ensure any simulation is sufficiently representative for the actual system, interface and the installation environment" [7].

Moreover, ARP4754 considers redundancy. It explains that "system architectural features, such as redundancy, monitoring, or partitioning, may be used to eliminate or contain the degree to which an item contributes to a specific failure condition." Moreover, "redundancy [provides] multiple implementations of a function either as multiple items, or multiple lanes within an item. It is a design technique based on the assumption that a given set of faults with the same system effect will not occur simultaneously in two or more independent elements. [...] The redundant elements may be parallel or backup, and their designs may be similar or dissimilar" [7].

ARP4761

ARP4761 defines redundancy as multiple independent means incorporated to accomplish a given function [8]. Here, selected partitions are configured redundantly. These redundant partitions provide the same functionalities as their counterparts and provide the same output when executed. In case of an application or partition fault, the faulty partitions are deactivated through a Health Monitoring (HM) partition. Their redundant counterpart is subsequently activated to ensure continued availability of the system.

ARP4761 directly discusses reconfiguration according to state changes: “A system changes state due to various events such as component failure, reconfiguration after detection of a failure, completion of repair, etc.” Moreover, “[w]hen considering fail safe events based on multiple failures, the analyst should consider contributions from incorrect outputs and inoperative protective or reconfiguration mechanisms” with a protective mechanism that is initially inoperative and engages once an element fails [8]. ARP4761 defines the events and states in the context of safety assessments, such as Error–A mistake in specification, design, or implementation; Fault–An undesired anomaly in an item or system; And Failure–A loss of function or a malfunction of a system or a part thereof [8].

For consistency, the definitions from ARP4761 for fault adaptation are adopted in the remaining paper.

B. Tools

This work utilizes MATLAB/Simulink to simulate dynamic reconfiguration in our avionics system. This is done with an add-on for MATLAB/Simulink called ARINC 653 Blockset. To get a realistic simulation environment, the PikeOS Certification Toolbox is leveraged for information about latencies, buffers, etc. that are taken from tests on embedded hardware. These tools are discussed in more detail in the following paragraphs.

ARINC 653 Blockset

The ARINC 653 Blockset utilizes Simulink to represent a subset of ARINC 653 services, enabling users to model, simulate, and generate code for an ARINC 653 partition within Simulink. The generated code adheres to the ARINC 653 Application Program Interfaces (API), making it suitable for compilation with an ARINC 653-compliant operating system.

In addition to the ARINC 653-specific blocks, the Blockset offers a utility that generates skeleton Simulink models from an ARINC 653 XML file. A skeleton model is created for each partition, and each model includes Simulink blocks that define processes within the partition. For this work, the Blockset is used as is with a bigger focus on its overarching simulation environment. Works showcasing the ARINC 653 Blockset with practical examples are [9] and [10].

PikeOS Certification Kit

PikeOS, the RTOS developed by SYSGO, is certified to high assurance levels like DO-178C DAL A, EN 50128 / 50657 SIL 4 or ISO 26262-6 ASIL D. To assist Original Equipment Manufacturers (OEMs) developing safety/security critical applications or systems, SYSGO provides a Certification Kit which is used to show conformity to certification standards towards certification authorities, mainly for avionics, industrial, automotive, or railway systems. The Certification Kit for PikeOS 5.1.3 contains requirement and interface documentation, test reports, and a set of analysis reports [11]. For an RTOS, mainly the worst-case execution times (WCETs) from timing analyses are considered. Mainly these are of interest when proving the execution completion of functions regarding their intended deadline. Besides the WCET, the timing analysis provides average execution times.

The time partition switch is special in that its timing analysis is determined by two main factors. These are divided into multiple execution paths which have different conditions to prolong the execution time. The two driving factors are: 1. `TPS_DURATION` –The duration of the actual time partition switch. This time is determined by different configuration elements and the current system state. 2. `TPS_PREV_DELAY` – The delay duration by which the time partition switch can be extended by a previous running partition due to actions which need to be completed before the actual switch can happen.

The reason for using PikeOS, along with its Certification Kit, as an exemplary operating system for timing analysis is its compliance with many of the standards previously introduced. Also, PikeOS is highly safety-critical and already used in many real systems, both in aviation as well as aerospace.

C. Definitions

The next paragraphs introduce definitions that are relevant for the simulation use case in this paper. Those are about technologies and concepts used in later parts of the paper.

Simulation with Prototyping

As defined in [12], a “prototype is a model implemented to evaluate a particular set of characteristics or attributes. The purpose of the [...] prototype is to ultimately serve as a standard by which its source requirements are judged. [...] [P]rototypes can be used early in the development of a system to identify and resolve validation issues.”

Typically, a full system development cycle eventually involves the testing of the system in its designated form. While testing on embedded hardware arguably provides the highest fidelity, that is, the most accurate specification towards the intended system, it comes with some drawbacks. The software integration, generation of artifacts, and deployment on the end system is time-consuming and, if done manually and not checked, prone to errors. Simulation with early prototypes can help overcome these drawbacks in early development phases. This implies, however, that the behavior and parameters from the intended system are sufficiently mapped to the simulation environment [13]. Figure 1 exemplifies the parts of a systems engineering process with the addition of a simulation environment for early prototyping. The simulation environment (depicted in green) contains as many quantitative parameters from the embedded hardware environment (depicted in gray) as required to achieve simulation with some desired accuracy. The green arrows indicate the rapid adjustment of software (architecture) parameters for further simulation iterations.

Dynamic Reconfiguration in the Context of ARINC 653

Dynamic reconfiguration in avionics refers to the process by which an avionics system adaptively modifies its hardware and software configurations during operation to respond to changing mission requirements or to recover from failures.

In IMA systems, dynamic reconfiguration allows for the redistribution of computational resources and the reassignment

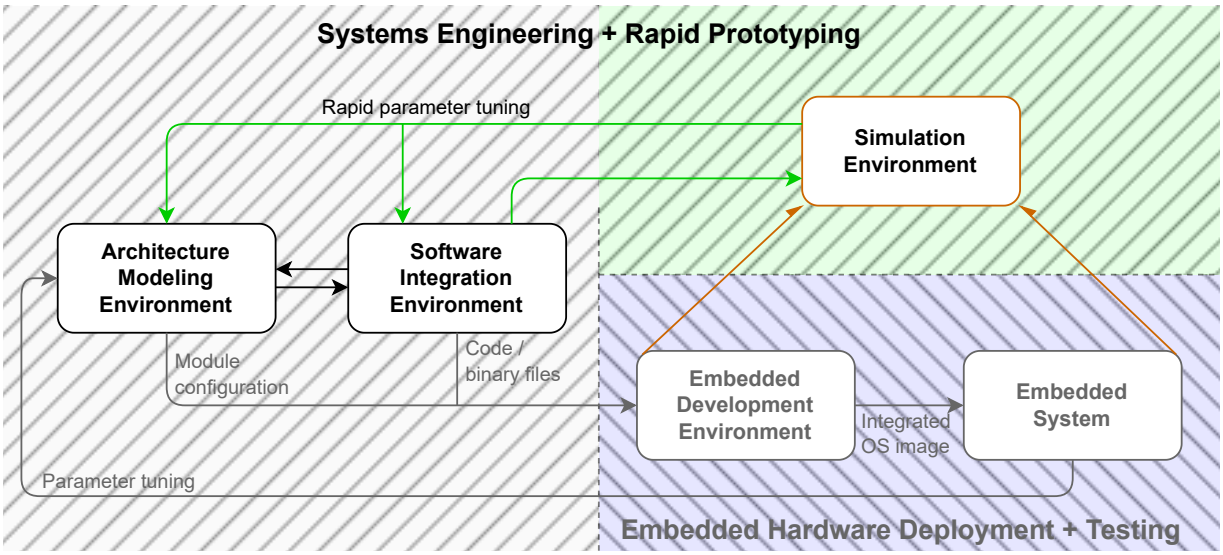

Fig. 1: Mapping of embedded system parameters to simulated environment for early testing of the desired system.

of functions among hardware or software components. This adaptability is particularly beneficial in maintaining critical functions during hardware or software failures, thereby improving overall system safety and availability. Bieber et al. showcase some concepts for dynamic reconfiguration with ARINC 653 in [3] which are further explained in [14].

According to ARINC 653, there are different error levels which need to be considered when performing fault management: Module level error, partition level error, and process level errors. Our work assumes either partition level errors or process level errors with error propagation to the partition level. Fault containment on module level involves more extensive system considerations, like a network backbone for module interfaces, which are out of scope for this work.

The paper at hand uses PikeOS as an exemplary hypervisor/RTOS to illustrate some of the concepts and methods. With PikeOS there are two major dynamic reconfiguration strategies, as shown in Fig. 2. These are the use of a dedicated HM partition or the multiple module schedule change. The two concepts are explained in more detail in the next paragraphs.

Health Monitoring and Partition Modes

The first option for dynamic reconfiguration encompasses a dedicated HM partition with elevated rights, which disables failed partitions and enables their redundant counterparts. HM partitions differ from conventional partitions, configured and scheduled for ARINC 653 avionics systems, as they receive information about failed partitions and perform some recovery action if desired. As described in ARINC 653, there is also the possibility to work with predefined fault containment responses, which are defined in the HM configuration tables. Together, these mechanisms form the following three HM fault management options:

- HM configuration tables to respond to faults with the OS
- Application partitions pass error data to the OS
- System partitions perform error management

The partition level HM enforces the partition level error recovery actions. The recovery action for partition errors is specified in the HM configuration tables. The recovery actions are as follows: Ignore, stop the partition (`IDLE`), restart the partition (`COLD_START` or `WARM_START`), or recovery actions which are OS dependent.

ARINC 653 specifies five HM services with dedicated APEX functions:

- `REPORT_APPLICATION_MESSAGE`
- `CREATE_ERROR_HANDLER`
- `GET_ERROR_STATUS`
- `RAISE_APPLICATION_ERROR`
- `CONFIGURE_ERROR_HANDLER`

The `GET_ERROR_STATUS` service provides the error code, the identifier of a faulty process, the address at which the error occurs, and the message associated with the fault. The `REPORT_APPLICATION_MESSAGE` service request allows the current partition to transmit a message to the HM function. `REPORT_APPLICATION_MESSAGE` may be used to record an event for logging purposes. The `RAISE_APPLICATION_ERROR` service request allows a running process to invoke the error handler process for the specific error code `APPLICATION_ERROR` .

In the demonstrator used for this paper, a dedicated HM partition is used. We therefore omit the `CREATE_ERROR_HANDLER` service with a custom error handler implementation in the HM partition. The `CONFIGURE_ERROR_HANDLER` service considers the use of multiple processor cores. We only assume single core systems, which is why this service is omitted as well. The aforementioned idling or cold starting of partitions is performed with a function that is not standard in ARINC 653, called `APEX_EXT_SET_PART_MODE` . This function is provided by PikeOS as an API extension for OS-specific features.

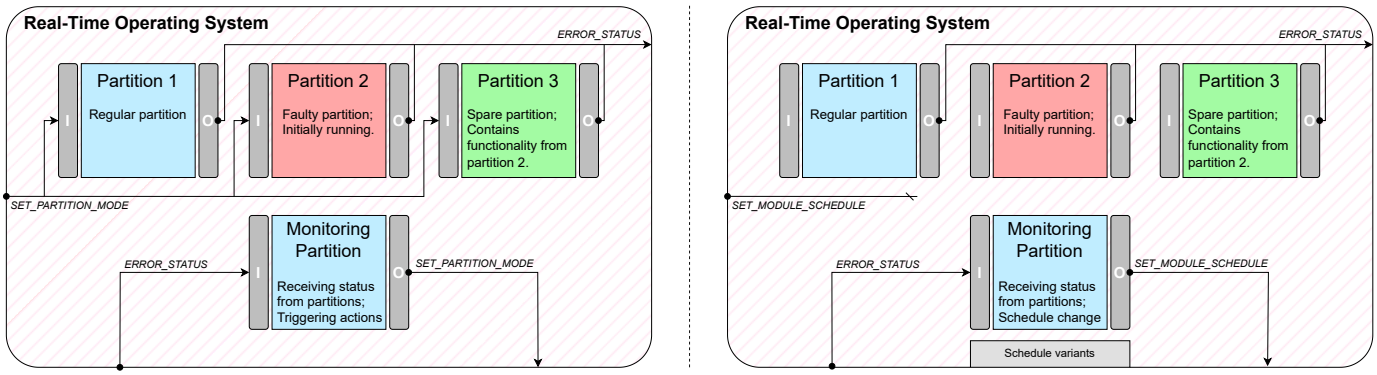

Fig. 2: Two different variations for dynamic reconfiguration supported by PikeOS.
Left: Reconfiguration with partition mode changes through Health Monitoring OS calls using elevated rights.
Right: Reconfiguration by triggering a schedule change for all partitions on the overall module level.

Multiple Module Schedules

The second dynamic reconfiguration strategy which is compliant with ARINC 653 is multiple module schedules. ARINC 653 Part 1 defines a single static module schedule. The module schedule is defined in the configuration table and repeats every major time frame of the processor's execution time. For some equipment, a single module schedule is too restrictive. One scenario where multiple module schedule is useful is component failures. In a cabinet architecture, programs critical to continuous flight could be allocated to run on another module (replacing a less critical application) when its primary module fails. In this case, a module schedule is selected to include applications from the failed module.

An authorized partition requests the OS to switch the currently running module schedule. The module schedule is switched at the end of the currently running module schedule. The partitions defined for the new module schedule will now run in place of those that ran with the old module schedule, i.e., with alternative partition time windows.

A partition is granted authority to set the module schedule by setting the `SetModuleSchedule` attribute for the partition in the configuration tables. The module schedule that will start executing at the end of the current module schedule, i.e., at the end of the current major time frame, is set with the function `SET_MODULE_SCHEDULE`.

While the multiple module schedules reconfiguration strategy could be considered as a dynamic reconfiguration strategy, this paper focuses on dynamic reconfiguration with a dedicated HM partition. Therefore, the multiple module schedules is disregarded.

III. Related Work

In their paper [1], Abdo et al. introduce a comprehensive framework designed to tackle the increased complexities associated with modern avionics systems. Their methodology, established within the AvioNET initiative, categorizes validation efforts into analytic model-, simulation-, and hardware-based approaches, promoting early detection of design issues to streamline the development process. However, the paper highlights certain limitations, such as the potential gaps between simulated outcomes and actual hardware performance, and it lacks extensive empirical evidence through case studies that could validate the framework's practical effectiveness.

In their paper [2], Brings et al. present a model-based prototyping approach aimed at enhancing the early validation of requirements for cyber-physical systems (CPS). The authors argue that traditional prototyping often leads to discrepancies between software prototypes and their intended hardware environments. To address this issue, they propose the derivation of an explicit prototype specification that maintains traceability between requirements and validation outcomes. However, a noted limitation is that their method may still require further developments in automated correction mechanisms and deeper integration with formal verification.

In their paper [15], Cui et al. explore various approaches to enhance the simulation and verification of IMA systems. One notable contribution is the proposal of a gray box simulation test environment, which leverages software fault injection technology to assess IMA software without delving into the complexities of avionics system testing. While this method offers a structured framework for testing, it is limited by its reliance on existing foreign technical standards and the challenges posed by data distribution service (DDS) technology.

Other notable works that discuss model-based analysis and simulation efforts specifically for dynamically reconfigurable avionics systems are [16], [17], and [18]. The paper at hand contributes to the collection of work regarding simulation of advanced avionics systems, such as those with dynamic reconfiguration capabilities, with some distinctions. First, the method considers software architectures for any architectures compliant with ARINC 653. The rapid prototyping is demonstrated with an ARINC 653-compliant hypervisor. Second, the simulation incorporates parameters, such as WCET, measured during hardware tests for that exemplary hypervisor. This approach demonstrates how to reach higher accuracy for simulation of complex systems. This enables further considerations for V&V and ultimately early certification for such safety-critical systems.

IV. Simulation for ARINC 653 Dynamic Reconfiguration Using a Prototype

We introduce an avionics system with an initial set of five partitions hosting one application each. In the next subsection, we explain the system configuration, parameters considered for the simulation environment, and the setup of the simulation environment. At the end of this section, we present an exemplary simulation for dynamic reconfiguration based on the predefined configuration and simulation parameter requirements for this system. One more simulation is performed with a more elaborate and complex system configuration. This approach provides some reference for the base simulation with added simulation coverage which increases the fidelity of the model validation. The system configuration is only introduced once for the first simulation. The second simulation is performed without prior elaborate system introduction.

A. System Configuration

The system configuration for this demonstrator is a standard configuration as described in ARINC 653 [4]. Such configurations are exemplified in [19, 20]. The applications from the demonstrator used for simulation are of varying Design Assurance Levels (DALs) and are scheduled at different frequencies, hereinafter defined in Hertz (Hz):

Application 1: Flight Control–40 Hz, DAL A
Application 2: Braking–500 Hz, DAL A
Application 3: Human Machine Interface–100 Hz, DAL B
Application 4: Terrain Awareness–10 Hz, DAL C
Application 5: Maintenance–1 Hz, DAL C

Three additional spare partitions are added to the initial set of five partitions. The spare partitions add redundancy by hosting one duplicate, i.e., similar, application each, specifically for *Flight Control*, *Human Machine Interface*, and *Braking*. In case these partitions enter a failure state, the respective redundant partitions take over functionality. The reconfiguration actions are performed by a dedicated HM partition. The essential configuration parameters for the different partitions are illustrated in Table I.

The applications are proprietary applications developed at CSIR National Aerospace Center (NAL) in Bangalore. The configuration parameters, such as the execution frequency for each partition, have been determined beforehand through requirements and general system considerations. The rationale behind the specific parameter values is not further explained here. The applications are modeled in MATLAB/Simulink and are used natively in the ARINC 653 Blockset environment. IMA currently does not host applications with high safety-criticality. The use case demonstrated in this paper shall contribute to an adoption of IMA for flight control functions by means of safety guarantees provided through simulation.

B. Simulation Parameter Considerations

We aim at increasing the fidelity of our simulation environment with V&V for certification considerations. Therefore, incorporating specific behaviors of the intended systems in the simulation environment is essential for performing realistic prototype tests. Our embedded end system hosts a *PowerPC VPX3-152 T2080* board with the PikeOS target build *ppc_e500mc-4g*.

From the existing proprietary runtime analysis provided by the PikeOS and ARINC 653 timing analysis, we get execution times for different functions and functionalities from PikeOS as well as the ARINC 653 APEX functions. Those are the ones introduced for the reconfiguration strategy explained in subsection II-C. The times are specified as WCET and Average Case Execution Time (ACET). The different functions with their WCET and ACET are shown in Table II.

As this paper makes considerations for early certification evidence, the simulation only considers WCET for simulation of functions and behaviors from the real system.

As mentioned before, the APEX services assumed for dynamic HM functionalities are `GET_ERROR_STATUS`, `REPORT_APPLICATION_MESSAGE`, as well as `RAISE_APPLICATION_ERROR`. These functions are regarded as one atomic subsystem, thereby combining their WCET for one HM action. For partition initialization, we only consider the `COLD_START` of partitions, as this gives us additional parameters to add to our simulation. These are the `CREATE_PROCESS`, `CREATE_QUEUING_PORT`, and `CREATE_SAMPLING_PORT` APEX services. `APEX_EXT_SET_PART_MODE` changes the modes of a partition, and the *Context Switch* describes the state change from one partition to another during runtime.

Parameters	Flight Control*	Braking*	Human Mach. Interface*	Terrain Awareness	Maintenance Functionality	Health Monitoring**
Period [ns]	25 000 000	2 000 000	10 000 000	100 000 000	1 000 000 000	N/A
Duration [ns]	1 000 000	1 000 000	1 000 000	1 000 000	1 000 000	Arbitrary
Periodical	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
Priority	1	2	3	4	5	0

TABLE I: Configuration of the HM partition and the five native ARINC 653 partitions hosting applications for this demonstrator.
*The redundant partitions share the same configuration parameters with their initially running counterparts.
**Configured with highest priority and executed whenever HM actions are required.

Timing Analysis	HEALTH MONITORING			APEX_EXT _SET _PART _MODE	PARTITION INITIALIZATION			Context Switch*
	GET _ERROR _STATUS	REPORT _APPL. _MESSAGE	RAISE _APPL. _ERROR		CREATE _PROCESS	CREATE _QUEU. _PORT	CREATE _SAMPL. _PORT	
ACET [ns]	2784	1430	1277	SOFTREAL	17 881	23 602	24 186	4988
WCET [ns]	6667	7921	8534	SOFTR.**	35 737	52 272	51 925	11 334

TABLE II: Timing analysis with execution times for the functions and behaviors considered as parameters in the simulation. *The timepartition switch is defined by a number of granular conditions like cache, thread, and synchronization operations. **Softreal timing behavior of $\mathcal{O}(F + M + C)$ dependent on F–number of open files, M–number of memory pools, and C–number of system extension threads registered callbacks.

The WCET for a context switch according to the PikeOS timing analysis is governed by multiple granular parameters, which are only explained on a high level here. They follow following calculation:

$$CS_{WCET} = \sum_{k=1}^{n_\alpha} TPS\_DURATION_{\alpha_k} + max(TPS\_PREV\_DELAY) + \Lambda. \quad (1)$$

Some of the parameters from equation 1 are explained in detail in the *PikeOS Certification Kit* paragraph of subsection II-B. There are some underlying conditions, such as:

$$\Lambda = configuredSystemtickDuration, \quad (2)$$

$$\alpha \ni (TPS\_DEFAULT,\ CACHE,\ THREADS,\ STRONG\_SYNC), \quad (3)$$

with each value in α governing the timepartition switch duration ($TPS\_DURATION$), depending on the system configuration. In this case, only the $TPS\_DEFAULT$, $THREADS$, and $CACHE$ are considered. The total WCET for the dynamic reconfiguration for one failed partition is therefore expressed as:

$$WCET_{Total} = HM_{WCET} + 2 \cdot SPM_{WCET} + PI_{WCET} + 2 \cdot CS_{WCET}, \quad (4)$$

with HM_{WCET} being the sum of WCET for all three APEX functions in HM and PI_{WCET} the sum of the three APEX functions for partition initialization. SPM_{WCET} describes the WCET for setting the partition mode and is factored in twice. Once for disabling a failed partition and once for enabling its redundant counterpart. The context switch CS_{WCET} is also factored in twice. Once for switching to the HM partition for reconfiguration actions and then to the redundant partition for initialization. Taking the values from the timing analysis illustrated in Table II, we get a $WCET_{Total}$ of 185 724 ns or ∼186 µs. This total assumes a negligible timing behavior for the `APEX_EXT_SET_PART_MODE` service as this service only has a softreal timing behavior.

C. Simulation Environment

The simulation environment is an extension of the ARINC 653 Blockset in MATLAB/Simulink. The extension encompasses functionalities for HM as defined in ARINC 653. More precisely, an additional partition for HM and management of partitions is added to the system setup. The HM partition hosts a state machine which performs the functionalities of an HM partition from the desired system. Figure 3 shows the setup in MATLAB/Simulink with the use of the ARINC 653 Blockset.

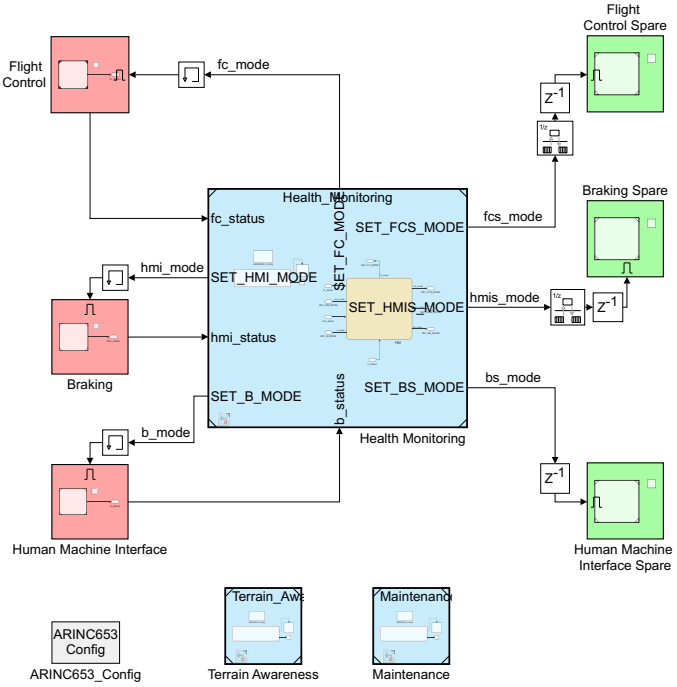

Fig. 3: Simulation model hosting a total of nine partitions.

The partitions *Flight Control*, *Human Machine Interface*, and *Braking* as well as their spare counterparts have interfaces with the HM partition. We argue that partitions with higher DAL levels, i.e., which are more safety-critical, need additional guarantees so that the operation of the aircraft is not affected in case of failures. Such guarantees can for instance be fault containment for those applications with redundant

partitions. The partitions *Terrain Awareness* and *Maintenance* have lower DAL levels and are therefore not considered for fault containment actions. Each partition which has a redundant counterpart sends its health status to the HM partition. In case of an error, which potentially leads to a subsequent failure, the HM partition decides on reconfiguration actions through the state machine logic. The state machine is shown in more detail in Fig. 4.

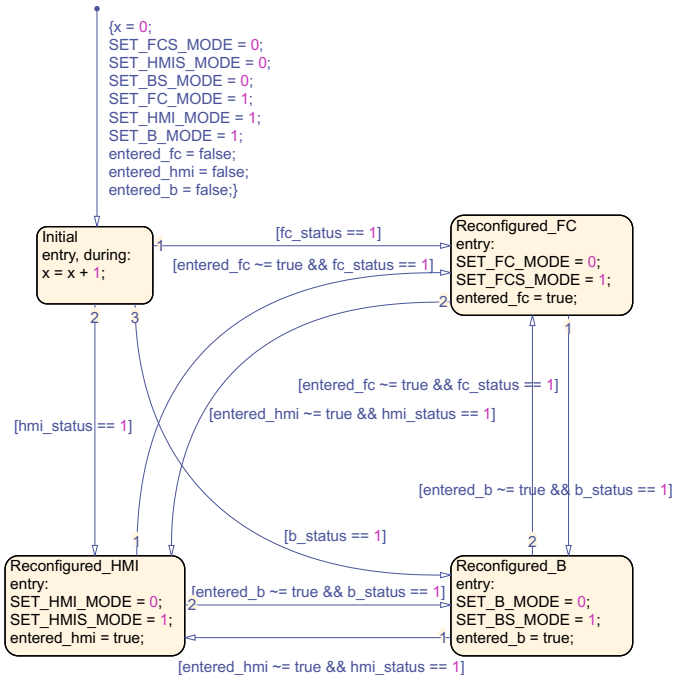

Fig. 4: State machine logic for dynamic reconfiguration.

After entering the initial state, the state machine can enter one of three reconfiguration states, one for each redundant partition. Each state can be entered only once. This means that reconfiguration is handled in a consecutive manner and that each failed partition is reconfigured once. Returning to previous states is not possible and we assume that if the spare partition fails as well, no further fault management strategies are deployed. Once the state machine receives the external trigger about an application or partition failure, it changes to a reconfiguration state for respective partition. Then, the mode of the failed partition is set to idle and its redundant counterpart is enabled by setting its partition mode to `COLD_START`.

The simulation assumes failure of all three safety-critical partitions. The failures are simulated with fault injections inside each partition block at random times throughout the simulation. The WCET of all functions relevant for reconfiguration as well as the context switches introduced above are modeled with delays during state change and reconfiguration action. Each partition contains a counter for tracking the cumulative number of executions over time during the dynamic reconfiguration scenario. The latencies from the additional overhead of the reconfiguration actions and context switches are factored in. The executions for the proposed reconfiguration scenario and the previously introduced system can then be plotted over time, as shown in Fig. 5.

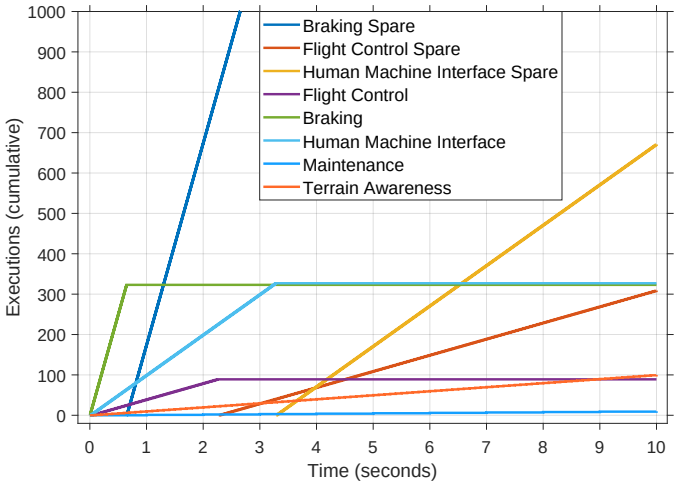

Fig. 5: Plot of cumulative executions for each application over a time period of 10 s.

The simulation is set to run for 10 s. The figure shows the execution of the initial five partitions, with some of them running into faults and subsequently failing. The values on the y-axis are limited to 1000 for better readability. All three partitions *Flight Control*, *Human Machine Interface*, and *Braking* were reconfigured with their spare counterparts *Flight Control Spare*, *Human Machine Interface Spare*, and *Braking Spare*. While the latencies from the reconfigurations are quite granular, there is a change in the number of completed executions in the simulation, as shown in Table III.

Application	Completed executions			Discrepancy*
	Regular Partition	Spare Partition	Total	
FC	90	310	400	0
B	324	4675	4999	1
HMI	328	672	1000	0
TA	100	N/A	100	0
MF	10	N/A	10	0

TABLE III: Executions of the five applications after simulation; broken down, where applicable, into executions from the initial (regular) partitions and their spares. FC–Flight Control, HMI–Human Machine Interface, B–Braking, TA–Terrain Awareness, MF–Maintenance Functionality.
*Difference between the total executions from this simulation and expected executions in a non-reconfiguration scenario.

These are calculated as the difference between the number of executions in a non-reconfiguration scenario and the executions of an application, including its spare implementation, in the simulated reconfiguration scenario. We observe one

execution miss for the *Braking* application which is involved in reconfiguration. The remaining applications, including the two partitions not involved in dynamic reconfiguration, do not show any deviation from the expected number of executions.

To validate the reliability of the simulation, an additional simulation is performed. This time, a total of 12 partitions are configured. All partitions except of partition eight are dynamically reconfigured during runtime and, similarly to the previous simulation, the discrepancy in application executions is calculated. The applications are all running with a duration of $2\,000\,000\,\mathrm{ns}$ and a periodicity of $24\,000\,000\,\mathrm{ns}$.

Application	Total completed executions	Discrepancy*
A1–A6	417	0
A7	416	1
A8	416	1
A9–A12	416	0

TABLE IV: Executions of 12 arbitrary applications; For simplicity, the executions are not broken down into those from regular partitions and their spares, unlike in Table III.
*Difference between the total executions from this simulation and expected executions in a non-reconfiguration scenario.

In this run, we observe one missed execution for applications seven and eight, each. This could be interpreted as peculiar behavior given that application eight is not directly involved in dynamic reconfiguration actions. The implications of this observation in the bigger context of both simulations runs are discussed in the next section.

V. Discussion

The difference in total executions for some applications in the simulation indicates overhead for dynamic reconfiguration actions. This is the anticipated behavior stemming from services and context switches from HM actions. Important to note is the low total latency from all services and behaviors involved in HM actions, as shown in Table II. The overall latency from one reconfiguration is in the range of a fraction of a ms. It is roughly a tenth of the smallest partition duration and periodicity, i.e., $2\,\mathrm{ms}$, as per the system configuration in Table I. Because of such low overhead, we only observe one missed execution for one application in the first simulation run. The second run, however, shows that further reconfiguration with a higher number of partitions can have a different effect on the overall system. Here, we observe two missed executions. One for a partition involved in dynamic reconfiguration but also for one partition not actively involved in dynamic reconfiguration. It seems that accumulated latencies or additional reconfiguration actions can have a cascading effect on the number of time partition window delays. The 11 reconfiguration actions from the 11 actively involved partitions result in a total latency that is higher than the common execution duration per partition. This leads to a delay significant enough to disrupt the initial schedule in a major frame, even affecting a partition not involved in dynamic reconfiguration. One more mechanism that needs to be taken into account is the varying added overhead depending on the time a reconfiguration action is performed in a failed partition. The *wastefully* consumed time from the faulty, not fully executed, partition will have a higher added latency with its reconfiguration happening towards the end of its time window. This effect also concerns partitions not actively involved in reconfiguration actions.

More generally, the simulation results indicate that the accuracy of the models used in this virtual V&V approach closely aligns with the expected behavior from the timing analyses performed on the target hardware. By mapping parameters derived from actual embedded systems, we achieved a higher level of confidence in the outcomes of the validation process.

VI. Conclusion

In an era of increasing complexity and demanding safety requirements for avionics systems, the necessity for robust virtual V&V methods becomes paramount. As highlighted in this paper, the simulation of a high fidelity prototype demonstrates a significant advancement in achieving accurate testing results for safety-critical systems like IMA. By adhering to established standards, including ARINC 653, DO-297/ED-124, and DO-331/ED-215, the proposed methodology not only aligns with industry expectations but also offers a structured framework for simulating dynamic system behaviors. The use of the ARINC 653 Blockset within MATLAB/Simulink facilitates an accurate representation of reconfiguration processes, ensuring that each simulated scenario remains compliant with system requirements.

Traditional HiL testing approaches face limitations in terms of cost, time, and resource allocation, especially amid the increasing complexity and safety requirements of modern avionics systems. Our methodology addresses these challenges by facilitating dynamic behaviors in a controlled simulation environment, thus enabling earlier intervention in the design process when issues arise. A positive observation from the results of the demonstration is that the simulation for the experiments aligns very well with the analysis obtained from reference tests on embedded hardware. This gives evidence for high fidelity in simulation and potential early certification considerations in the MBD process.

However, while the demonstration shows promising results, some considerations must be acknowledged. For instance, the simulation introduces potential gaps between simulated environments and actual hardware performance. To tackle this shortcoming, the simulation should be verified against its intended system. For this, tests on the actual hardware, in this case a *PowerPC VPX3-152 T2080* board, shall be conducted to validate potential guarantees provided by the simulation. As the work at hand is more of a feasibility study for virtual V&V activities with simulation, potential shortcomings will be addressed in future work.

VII. Acknowledgments

The authors thank the associates Juan Valverde, Marco Bimbi, Stefan Raab, Mark McBroom, and Jan-Cornelius vom Hau from The MathWorks who provided insight and expertise that assisted the research. Their contributions in the field of dynamic reconfiguration and avionics engineering with MATLAB/Simulink greatly facilitated this work. Moreover, the Certification Kit provided by SYSGO along with their expertise in runtime analysis leveraged the simulations discussed in this paper.

This work is part of a project that has received funding from the Federal Ministry for Economic Affairs and Climate Protection (BMWK) of Germany as part of the aviation research program LuFo VI-2 under Grant Agreement No 20Y2105 – WISDOM: Wing Integrated Systems Demonstration On Mechatronic Rig.

References

[1] Kamiran Abdo, Jasmin Broehan, and Frank Thielecke. "A Seamless and End-to-End Approach for Early and Continuous Validation of Next-Generation Avionics Platforms". In: *Software Engineering 2023 Workshops*. Bonn: Gesellschaft für Informatik e.V., 2023, pp. 130–149. DOI: 10.18420/se2023-ws-15.

[2] Jennifer Brings et al. "Model-Based Prototype Development to Support Early Validation of Cyber-Physical System Specifications". In: *RESACS: Second International Workshop on Requirements Engineering for Self-Adaptive and Cyber Physical Systems*. Mar. 2016. URL: https://ceur-ws.org/Vol-1564/paper15.pdf.

[3] Pierre Bieber et al. "Preliminary design of future reconfigurable IMA platforms". In: *SIGBED Rev.* 6.3 (Oct. 2009). DOI: 10.1145/1851340.1851349.

[4] ARINC. *Avionics Application Software Standard Interface*. Standard. Bowie, MD, USA: Aeronautical Radio Incorporated, Dec. 2019.

[5] RTCA. *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*. Standard. Washington, D.C., USA: RTCA, Nov. 2005.

[6] RTCA. *Model Based Development and Verification*. Standard. Malakoff, France: RTCA, Jan. 2012.

[7] S-18 Aircraft and Sys Dev and Safety Assessment Committee. *Guidelines for Development of Civil Aircraft and Systems*. SAE International, Dec. 2010. DOI: 10.4271/ARP4754A.

[8] S-18 Aircraft, Sys Dev, and Safety Assessment Committee. *GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT*. SAE International, Dec. 1996. DOI: 10.4271/ARP4761.

[9] Gianluca Corraro et al. "A novel approach for the development and coding of avionics functionalities for IMA architectures". In: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. 2018, pp. 1–8. DOI: 10.1109/DASC.2018.8569824.

[10] Bojan Lukić et al. "From Architecture Models to a Target Platform: A Systematic Approach to Model-Driven Development of Dynamically Reconfigurable Avionics Systems". In: *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*. 2024, pp. 1–10. DOI: 10.1109/DASC62030.2024.10749454.

[11] SYSGO GMBH. *PikeOS with Certification Kits for the highest Safety Levels*. Standard. Klein-Winternheim, Germany: SYSGO GMBH, Feb. 2021.

[12] B.A. Andrews and W.C. Goeddel. "Using rapid prototypes for early requirements validation". In: *AIAA/IEEE Digital Avionics Systems Conference. 13th DASC*. 1994, pp. 70–75. DOI: 10.1109/DASC.1994.369501.

[13] Umut Durak, Andrea D'Ambrogio, and Paolo Bocciarelli. "Safety-critical simulation engineering". In: *Proceedings of the 2020 Summer Simulation Conference*. SummerSim '20. San Diego, CA, USA: Society for Computer Simulation International, 2020. ISBN: 9781713814290. URL: https://dl.acm.org/doi/abs/10.5555/3427510.3427535.

[14] Bojan Lukić et al. "State-of-the-Art Technologies for Integrated Modular Avionics and the Way Ahead". In: *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. 2023, pp. 1–10. DOI: 10.1109/DASC58513.2023.10311229.

[15] Xining Cui, Yukai Hao, and Xiaodi Dai. "Simulation Verification Technology of Virtual-Real-Integration Avionics System". In: *7th International Conference on Dependable Systems and Their Applications (DSA)*. 2020, pp. 472–477. DOI: 10.1109/DSA51864.2020.00080.

[16] Zeyong Jiang et al. "New Model-Based Analysis Method with Multiple Constraints for Integrated Modular Avionics Dynamic Reconfiguration Process". In: *Processes* 8.5 (2020). ISSN: 2227-9717. DOI: 10.3390/pr8050574.

[17] Quan Zhang, Shihai Wang, and Bin Liu. "Approach for integrated modular avionics reconfiguration modelling and reliability analysis based on AADL". In: *IET Software* 10.1 (2016), pp. 18–25. DOI: 10.1049/iet-sen.2014.0179.

[18] Dajiang Suo, Jinxia An, and Jihong Zhu. "AADL-based Modeling and TPN-based Verification of Reconfiguration in Integrated Modular Avionics". In: *18th Asia-Pacific Software Engineering Conference*. 2011, pp. 266–273. DOI: 10.1109/APSEC.2011.27.

[19] Bojan Lukić et al. "Automated Configuration of ARINC 653-Compliant Avionics Architectures". In: *AIAA SCITECH 2024 Forum*. 2024. DOI: 10.2514/6.2024-1856.

[20] Bojan Lukić, Sven Friedrich, and Umut Durak. "A Streamlined Approach Toward Automated Generation and Validation of ARINC 653–Compliant Avionics Configurations". In: *Journal of Aerospace Information Systems* 22.5 (2025), pp. 314–324. DOI: 10.2514/1.I011439.