https://doi.org/10.1590/jatm.v17.1381 ORIGINAL PAPER

A Virtualized Architecture for Software-in-the-Loop Testing Applied to the LEON3 Processor

Luiz Henrique Antoniassi Santos[1,*], Jackson Tavares Veiga[1], Rodrigo de Marca França[1], Fabio Rofino[1], Carlo Terzaghi Tuck Schneider[1], Marco Antonio Furlan de Souza[1], Sergio Ribeiro Augusto[1], Daniel Dalla Vecchia Gueter[1], Vanderlei Cunha Parro[1]

1. Instituto Mauá de Tecnologia – Núcleo de Sistemas Eletrônicos Embarcados – São Caetano do Sul/SP – Brazil.

***Corresponding author:** luiz.antoniassi7@gmail.com

ABSTRACT

The increasing complexity of embedded systems in aerospace missions, particularly within the New Space paradigm, calls for more agile and cost-effective approaches to software and hardware integration. Traditional prototype-heavy development cycles are being replaced by virtualization and emulation techniques that support faster, iterative validation. Despite the growing adoption of such techniques, few studies propose a flexible and stable software-in-the-loop (SIL) framework tailored to emulated environments in the aerospace sector, especially considering open-source and widespread tools and technologies. This work addresses this gap by introducing a formal and adaptable SIL testing architecture based on the Quick EMUlator (QEMU), an open-source emulation platform, as its core. The framework targets the LEON3 processor, widely used in aerospace applications, and was validated through three sequential test scenarios integrating emulated environments and physical counterparts. These tests assessed software correctness, logical consistency, and timing behavior. Results confirmed full test success rates and the logical fidelity of the virtualized system, while revealing inherent timing discrepancies, characterized by an average advance of 20 ms in processing and transmission times compared to the physical counterpart. Despite these differences, the framework demonstrated sufficient accuracy and reliability for software testing in virtualized environments, provided its timing variations are properly accounted for.

Keywords: Aerospace systems; Virtual properties; Software development tools; Software engineering; Computer systems simulation; Spacecraft electronic equipment.

INTRODUCTION

The space age has undergone a significant transformation, moving from the rigid and linear methodologies of Old Space to the more agile and adaptive approach known as New Space (Golkar and Salado 2020; Peeters 2024; 2021). This shift is being driven by significant private and government investment that is reshaping the space innovation and technology sectors (Paikowsky 2017; Peeters 2024).

This new paradigm is driven by the increasing complexity of embedded systems in satellites and other aerospace missions, which demand greater flexibility and efficiency in the development and integration of software and hardware (Kanavouras *et al.* 2022). This aspect also increases the challenges in methodologies associated with the mission's development lifecycles (Khurana and Hodges 2020; Kopetz and Steiner 2022; Marwedel 2021).

Received: Sept. 06, 2024 | **Accepted:** Mar. 31, 2025
Peer Review History: Single Blind Peer Review.
Section editor: Paulo Renato Silva

In the Old Space approach, extensive use of hardware prototypes is considered (Kanavouras *et al.* 2022), which increases the intricacy of development and therefore requires more validation processes. To effectively support a shift to agile methods, research is needed to develop means to reduce prototyping steps and to consider the use of software for extensive testing and validation of components in sophisticated systems (Khurana and Hodges 2020).

The use of emulators is one strategy to eliminate some of the prototyping phases and has become essential to meet the challenges of testing and validating critical components, such as processing units (Choi and Nam 2012; Gutierrez *et al.* 2023), without incurring the high costs associated with traditional approaches (Buck *et al.* 1994; Teich 2012).

These practices not only accelerate system creation, but also improve component integration, which is essential for the success of complex space missions (Mihalič *et al.* 2022; Weltzin and Delgado 2009). For example, studies have shown that simulation of hardware and software using tools such as Quick EMUlator (QEMU) allows the emulation of embedded controllers, simplifying the tests and verification processes (Bekele *et al.* 2023; Gutierrez *et al.* 2023). More specifically, QEMU is an open-source tool that enables virtualization and emulation across a broad range of architectures and subsystems – including widely used ones such as ARM and x86, as well as those specifically targeted at embedded systems, such as RISC-V and SPARC.

Thus, the proposal of this work aims to establish a formal structure for testing software using the software-in-the-loop (SIL) methodology applied to emulated systems. This attempts to fill a gap in the existing literature by offering a structured approach to SIL testing and verification for embedded systems, with emulation techniques serving as the primary tool. The research outline involves the implementation of the proposed SIL architecture into a framework utilizing QEMU as a hardware emulator and the LEON3 processor (Sturesson *et al.* 2011), widely used in the aerospace context, as the main software test platform. Then, the main goal is to test software functionalities based on an emulated system through a SIL test framework, further comparing it to a corresponding physical environment and assessing its accuracy.

The fidelity and accuracy of the emulated SIL architecture can be parameterized by temporal metrics, such as processing and data transmission timings, as well as logical ones, such as the success rates of the algorithm execution in each test run. Thus, its accuracy relative to the real hardware implementation can be empirically assessed by evaluating its impact across different software testing scenarios, such as embedded control systems or telecommands (TCs) and telemetry (TMs) transmission in telecommunication systems, for instance.

Thus, the specific objectives guiding the proposed work are:

- Identify gaps in the literature and provide a solid theoretical basis for the proposed architecture.
- Propose and describe an architecture for emulated testing, detailing its components and structure.
- Implement the SIL structure into a framework, focusing on configuring its setup and systematizing test procedures.
- Test and validate the framework under a SIL test, emphasizing logical and temporal performance.
- Evaluate the effectiveness of the developed model in different scenarios and test cases, by comparing the emulated SIL tests with equivalent ones based on real hardware.
- Provide a clear understanding of the results obtained and evaluate the operational efficiency of the model under test conditions.
- Reflect on the testing framework's effectiveness and accuracy and its potential impact on the aerospace sector, identifying its strong and weak points across different software testing scenarios, based on temporal and logical metrics.
- Propose recommendations for future research on the defined testing architecture, with the aim of, among other objectives, enhancing its fidelity to physical testing platforms across a broad range of cases.

The remainder of this paper is organized as follows: the rest of this introduction includes a bibliographic review that outlines key concepts, existing solutions, and defines the scope of the proposal; the methodology section is divided into two main subsections, the proposal subsection introduces the proposed architecture for emulated testing, detailing its structure and components, and the implementation subsection describes the SIL setup and the systematization of testing procedures; the tests and results section presents unit test cases using decision tables and provides a statistical analysis of the test outcomes; finally, the conclusion summarizes the findings and discusses the applicability and impact of the proposed architecture and implemented framework.

BIBLIOGRAPHIC REVIEW

This subsection has two main goals: first, to present an overview of the key concepts reviewed in this study that inform the definition of the architecture; second, to provide clarity on the technical terminology and concepts that are consistently referenced throughout the article.

Related concepts

Several simulation and emulation methods are used to replicate embedded circuits to ensure system efficiency and fidelity. Among these approaches, the following stand out:

• Co-simulators: it is a technique that integrates with different types of models and systems to analyze their behavior. Instead of simulating isolated components, co-simulation allows different elements or parts of the system to be replicated together. A promising study was presented in Biagetti *et al.* (2023), which addresses hardware and software co-simulators as an essential technique for designing and validating embedded circuits. This work introduced solutions to simulate field-programmable gate array (FPGA) projects in conjunction with controllers. However, co-simulation environments require complex tasks, demanding different levels of digital model architecture (Díaz *et al.* 2021).

• Hardware-in-the-loop (HIL): is a testing process in which a plant is executed on a real-time simulator and interacts with a physical controller or rapid control prototype. This approach is useful for incremental prototyping of complex architectures, where parts of the system may be physical components that interact with the simulator (Mihalič *et al.* 2022; Mina *et al.* 2016).

• SIL: it is a methodology that involves testing software in a virtual environment that simulates the behavior of both the target hardware and external factors. This approach reduces costs and enables early detection of issues and bugs, making it crucial in critical applications like aerospace, particularly in satellite development (Kiesbye *et al.* 2019).

• Processor-in-the-loop (PIL): it is a test method that allows designers to evaluate a controller within a specific processor environment (Mina *et al.* 2016). Processor-in-the-loop can also be useful for testing components of a complex system, which can be integrated into a digital platform. The method facilitates debugging both the controller and the subsystem under test, allowing the correction of potential performance problems.

To better comprehend the different simulation approaches presented, refer to Table 1.

Related to the previous context exposed, a well-established emulation equipment for testing in the aerospace sector is the Simucam (Ferrão *et al.* 2012; 2016), a simulator developed by the Núcleo de Sistemas Eletrônicos Embarcados of the Instituto Mauá de Tecnologia (NSEE-IMT) in collaboration with the European Space Agency (ESA). This equipment was explicitly designed to provide real-time emulation of the cameras of the PLATO mission (ESA 2017) and to support the development and validation of

Table 1. Direct comparison of the presented simulation and emulation methods, highlighting their limitations and advantages.

Method	Description	Key advantages	Limitations	References
Co-simulation	Simulates hardware and software components together for integrated testing	Testing at multiple levels Verification of interactions between components	High computational resources Complex setup	Biagetti *et al.* (2023), Díaz *et al.* (2021)
HIL	Uses a real physical controller interfacing with a simulated environment	Real-world testing conditions Compatibility with physical components	Access to hardware required Costly and time-consuming	Mihalič *et al.* (2022), Mina *et al.* (2016)
SIL	Tests software in a fully virtualized environment that simulates the physical system	Reduced costs and setup complexity Early-stage validation	Timing discrepancies May lack real-world hardware behavior	Kiesbye *et al.* (2019)
PIL	Runs the controller software on the actual processor while keeping other components simulated	Better accuracy than SIL Processor-specific performance evaluation	Dependent on specific processor architecture Potential timing issues	Mina *et al.* (2016)

Source: Elaborated by the authors.

some of the mission subsystems. While such hardware-based solutions are effective, they have inherent limitations. Some of these are the need to create specific components, which adds cost and time to development, being tied to a very specific end platform.

In contrast with hardware-based simulators such as the Simucam, this study reviewed the major hardware emulation tools available as an alternative:

- Gem5: is a microservice architecture that models entire systems but has limitations when emulating various devices. It significantly reduces emulation speed at the architecture's limit, especially when handling multiple protocols or processors working in parallel in the virtual hardware environment (Abudaqa *et al.* 2018, Binkert *et al.* 2011).
- Open Virtual Platforms (OVP): is offered by OPVSIM, which acts as an instruction set simulator (ISS), but does not provide cycle-accurate simulation. The paper of Lonardi *et al.* (2014) discussed how modules work consistently across QEMU and OVP environments.
- QEMU: is an open-source software emulator widely utilized across various hardware processor architectures, including x86, ARM, MIPS, RISC, and SPARC (SPARC 1994). By employing dynamic binary translation (DBT) (Probst 2002), QEMU executes binary code at high speeds, making it well-suited for advanced system emulation. Despite its strengths, QEMU has limitations in handling multithreaded processing for multiple tasks, which presents opportunities for research in emulating embedded systems (Bekele *et al.* 2023; Choi and Nam 2012; Díaz *et al.* 2021; Gutierrez *et al.* 2023; Ziemke *et al.* 2011).

A comparison of software offering embedded systems emulation capabilities is summarized in Table 2.

Table 2. Direct comparison of various available emulation tools, especially for embedded systems, and their aspects.

Emulator	Key features	References
Gem5	Microarchitecture Cycle-accurate simulations Full-system mode supporting Limitations with multiple protocols Lower simulation speed compared to QEMU and OPVSim	Abudaqa *et al.* (2018), Binkert *et al.* (2011)
OVP	Private virtual platform emulator Simulation of multiprocessor platforms Instruction-accurate, not cycle-accurate Extensive documentation Limited support for hardware processors	Lonardi *et al.* (2014)
QEMU	Open-source software emulator Support for various processors architectures DBT for high-speed executions Limitations in multithreaded processing Widely used in satellites systems virtualization Support for SPARC architecture and LEON3 processors	Charif *et al.* (2019), Mina *et al.* (2016), Rodrigues *et al.* (2022), Choi and Nam (2012)

Source: Elaborated by the authors.

Ziemke *et al.* (2011) used QEMU within simulation frameworks for small satellite development, while Choi and Nam (2012) demonstrate its application in emulating aerospace processors, highlighting its relevance in this field. Quick EMUlator (QEMU) supports several ARM systems and already offers emulation support for the LEON3 processor, broadly used in aerospace applications. These features underscore its significance in system emulation. Nonetheless, there are significant opportunities for advancing the development of increasingly complex and flexible emulated systems within the aerospace sector, particularly with QEMU. These opportunities motivate the creation of a framework for space ecosystems, focusing on integrating and validating QEMU with the LEON3 processor. The widespread use of QEMU in various high-level system applications and its open-source nature makes it a valuable tool.

This review highlights the essential role of these technologies and their applications, especially in the aerospace scope. The next section will reinforce the existing solutions focusing on delimiting the proposal.

Existing solutions

This section of the literature review aims to examine established works that highlight the use of emulated systems for conducting and facilitating software testing, thereby supporting the proposal of this study.

Díaz *et al.* (2021) presented an overview of the different platforms for virtualization and simulation of hardware with software. Quick EMUlator (QEMU) was mentioned, with adaptations to parallelized simulation, enhancing the efficiency of testing and validation of complex embedded systems.

The ARCHIE tool utilized QEMU to simulate various types of failures in embedded devices, generating automated tests using binaries that emulated the desired hardware conditions (Hauschild *et al.* 2021).

In Illescas (2022), the use of simulators to emulate both SPARC and PowerPC architectures was discussed. In this context, QEMU was adapted to trace instructions, facilitating performance verification.

White and Pilbeam (2010) studied and assessed the limitations of various virtualization techniques across different cases and tools, evaluating the performance of virtualization for modern multi-core CPU architectures using KVM-QEMU.

A promising study on hardware and software co-simulators as a unified technology for circuit design and validation is presented in Mina *et al.* (2016). This work introduced methods for simulating FPGA projects alongside controllers and developed an open-source solution to address the design and verification needs of embedded circuits.

On the other hand, Gutierrez *et al.* (2023) explored the use of QEMU in the context of microcontrollers, emphasizing emulation as a cost-effective and rapid solution for replicating the functionality of physical systems, particularly embedded controllers.

Bekele *et al.* (2023) highlight the use of QEMU in the development of robustness testing techniques for space systems. This approach allows the identification and correction of potential failures before real-world testing. The study also describes the use of QEMU as a robust simulation environment for fault injection, where it has been used to test and validate systems under adverse conditions, like those experienced by satellites in orbit.

Finally, Cordero *et al.* (2011) aim to create a Software Development and Validation Facility (SDVF), designed to support the lifecycle of spacecraft equipment containing on-board software (OBSW) while maintaining low implementation costs. The work utilizes simulation tools provided by Gaisler Research to simulate LEON processors, providing a foundation for software testing.

While some studies use QEMU for various aspects of emulation, such as failure simulation, performance verification, and cost-effective solutions, there is a lack of research that establishes the tool into a formalized test architecture based on a SIL methodology. By emulating embedded system behavior in a controlled and deterministic environment, the SIL methodology emerges as one of the most practical and effective approaches for embedded software testing. While primarily focused on software validation, SIL also enables flexible configuration of the emulated hardware, streamlining implementation and facilitating the evaluation of code and algorithms across diverse scenarios. In this context, the study proposes a comprehensive SIL testing structure for aerospace embedded systems using QEMU as a hardware emulator.

The next section contains a detailed description of the scope of the proposal, especially in terms of the main aspects that define the design of the SIL testing architecture.

Scope definition

The proposed scope has grouped the concepts presented in the bibliographic review, creating a simplified model that satisfies the requirements of the project. Each of the proposal's components is explained in the items below:

- Agile development: this serves as the primary motivation for the proposed work, positioning emulation tools for SIL testing as essential for continuous hardware and software development techniques.
- SIL methodology: is widely used to test and validate software in an environment that simulates the physical system, with the aim of identifying and correcting errors before its implementation on real hardware.
- QEMU emulator: it has previously been used in aerospace applications and the projects studied demonstrated its validity in emulating processes and simulating small satellites. The key factors considered for utilizing QEMU in the proposal were: it is an open-source software; it supports to SPARC architecture processors, such as the LEON3; and has the potential ability to develop robust frameworks for space ecosystems.
- LEON3 processor: is the main hardware platform chosen to run the tests, as it represents the general processing core of the control unit in the final equipment of a space mission. The choice of this processor is due to its reliability and wide acceptance within the aerospace community, as mentioned in the literature review.

• External test entity: an external test unit was essential to perform comprehensive and effective SIL-based testing. The code acts as the main test entity and is strategically placed at the heart of the test setup. Its primary objective is to orchestrate sequential tests according to the SIL methodology for each connected system, providing stimuli, receiving responses and analyzing them.

The next section provides a detailed description of this work's proposal and its implementation, including the establishment of an architecture for SIL testing on virtualized hardware, specifically analogous to the LEON3 processor.

METHODOLOGY

Proposal

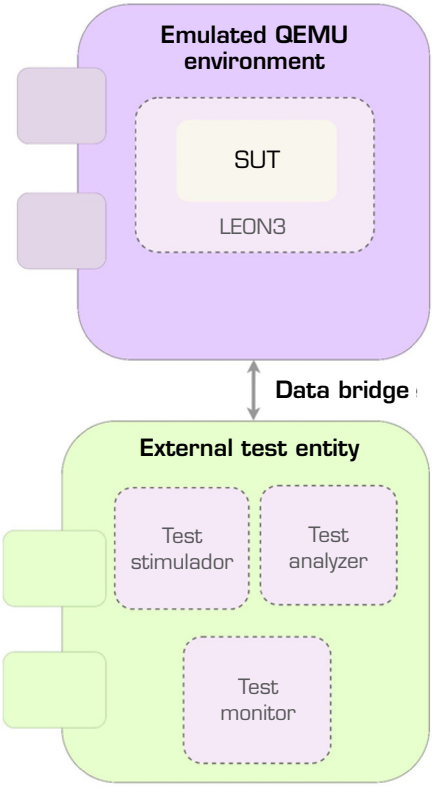
This work aimed to implement a SIL testing architecture using emulated systems, particularly centered on the LEON3 processor. The proposal also includes its implementation, which is carried out using several tools, particularly QEMU.

In the shown context, the structural diagram of the SIL framework architecture was established, as illustrated in Fig. 1. Two main components can be observed: the emulated environment and the external test entity.

In addition to exposing communication interfaces to external parts, the emulated environment is responsible for instantiating the emulated components alongside the system's main processor. It also executes the software, acting as the system-under-test (SUT), on the virtualized processor, which manages other peripherals common to the environment.

The external test entity has three main functionalities, represented by different software components:

• Test stimulator: generates stimuli for the emulated QEMU environment.
• Test monitor: monitors the received test responses from the emulated environment.
• Test analyzer: analyzes the test responses, comparing them to the provided stimuli.



Source: Elaborated by the authors.

Figure 1. Theoretical representation of the SIL architecture of the proposal.

A data bridge can be established between both environments to facilitate the data transmission/reception necessary for SIL testing. Depending on the case, the data transmission bridge can be implemented using different communication protocols (such as UART, TCP/IP, etc.), based on the requirements of the SUT.

The components of the testing entity, established on the chosen communication protocol for the SUT, can be organized into a structure designed for unit testing of the embedded code. In this setup, the code's behavior is evaluated by applying specific inputs and comparing the results with expected values.

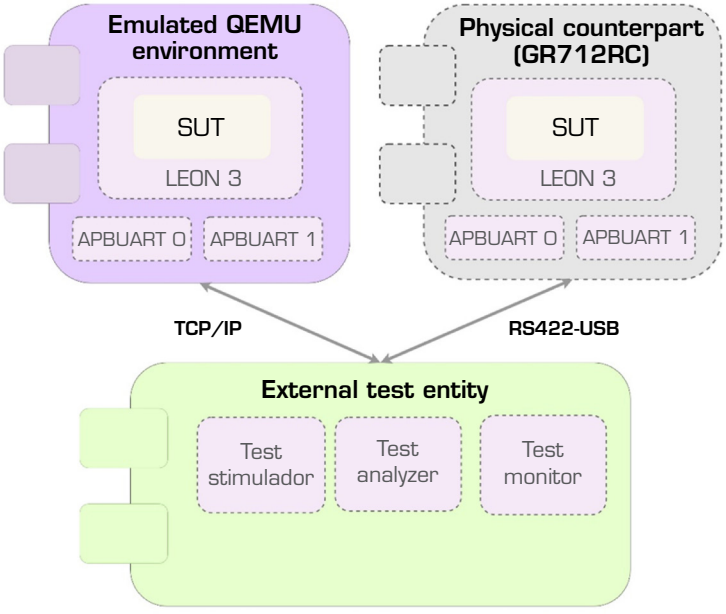
Sequential tests can also be performed to verify the robustness of the SUT under a higher load. In that case, the three components of the external test entity would be organized to sequence a predetermined number of consecutive unit tests, varying the inputs used in each one.

To implement the proposed architecture in a concrete form, a simple algorithm can be selected as the SUT: the Caesar Cipher, a deterministic encoding method that shifts each letter of the original message by a predefined value within an arbitrary sequence, such as the alphabet. Additionally, the external test entity can be implemented in Python due to its ease of use and efficiency. Furthermore, communication between the test entity and the emulated system can be established via the TCP/IP protocol, which QEMU supports as an external interface.

Finally, the proposed model will undergo multiple testing and verification scenarios after its implementation. Limitations in emulator usage across different contexts and architectures are well-documented, such as those regarding execution speed discrepancies compared to physical counterparts (White and Pilbeam 2010). From this perspective, the feasibility and constraints of the framework will be assessed, with a focus on the virtualized system's ability to accurately represent physical embedded hardware, particularly in the context of providing efficient environments for SIL testing, while considering both temporal and logical standards. The next subsection details the integration of the architecture into a cohesive framework, incorporating the previously mentioned tools and algorithms.

Implementation

The implementation of the proposal was carried out using several tools previously suggested. The overall structure of the SIL testing framework is illustrated in Fig. 2.



Source: Elaborated by the authors.

Figure 2. Diagram of the complete framework setup, encompassing both the virtualized LEON3 system and the actual LEON3 hardware.

The integrated proposed system is structured as follows:

- External test entity: positioned at the core of the setup, this module serves as the external test entity in the proposed framework, managing the validation of the selected software according to the SIL methodology. It was developed using standard Python, enhanced with the pySerial library (Liechti 2015) to enable serial communication. Also, it establishes an interface between the emulated and physically implemented systems, ensuring seamless testing across both environments.
- GR712RC board: as a counterpart to the emulated system and for validation purposes, a GR712RC development board from Frontgrade Gaisler (Habinc *et al.* 2010) was used, which can be seen on the right of Fig. 2. This board features two LEON3 cores and six available APBUARTs, which are simple UARTs connected to the system's APB bus, of which two were employed for serial communication with other components.
- Emulated QEMU environment: on the left of Fig. 2, it utilizes the LEON3 processor as its central unit. In this setup, pre-compiled software to be tested is allocated. Also, the emulated system instantiates two APBUARTs, allowing for external communication with other entities.

The LEON3 core in the emulated QEMU environment communicates with the external test entity via a TCP/IP connection. Both APBUARTs instantiated by the emulated QEMU environment provide TCP/IP links, which are connected to the test entity. On the real counterpart of the emulated environment, the LEON3 core communicates with the test entity through serial communication, mediated by an RS422-to-USB adapter that connects the two APBUARTs on the GR712RC to the operating system running the external test entity.

Finally, an embedded code was developed to run on the LEON3. In addition to implementing the Caesar Cipher algorithm described in the proposal, this code manages the APBUARTs for data transmission and reception in each system where it is executed. Thus, this code serves as the SUT for both environments.

Each component of the adopted structure will be detailed in the following subsections. The data flow in each unit test orchestrated by the test entity, which constitutes the main contribution of this work, will be shown as well.

External test entity

As the main scientific contribution of the proposal, the external test entity was responsible for:

- Controlling communication between the connected APBUARTs.
- Injecting stimuli through communication channels into the algorithms to be tested.
- Receiving and verifying responses produced by the algorithms.
- Timestamping the processing and transmission times of each test cycle.

It consolidated the roles of stimulator, monitor, and analyzer of the entity described in the proposal into a single system. The implemented entity is also responsible for all aspects related to the user interface, such as the input of test configurations and presentation of relevant statistical measures to the user, such as the standard deviation and average of the recorded transmission, processing and receipt times.

As the main control unit of the test, the role of the test entity will be better understood in the explanation of the data flow of a complete SIL test running on the framework.

GR712RC board

Since the goal of the proposal is to validate SIL from the perspective of hardware emulation for the LEON3 processor, it was necessary to use a physical reference to enable comparison with the validation test cycles performed on the equivalent emulated system. Thus, the GR712RC development board from Frontgrade Gaisler was selected, which in addition to having the two LEON3 cores required for the tests, also provided additional peripherals that could be used as needed.

The proposal encapsulated devices that were needed to provide communication interfaces between the hardware and external systems. Among other more advanced peripherals, UARTs were selected due to their ease of handling and control. Furthermore, since the software tests to be performed did not require high data transmission rates, the choice of UARTs was the most suitable.

It was observed that the development board features six UARTs, all connected via the APB bus to the LEON3 cores and thus referred to as APBUARTs. Two of these devices allowed serial communication via RS232 or RS422 interfaces, while the remaining four communicated exclusively through the RS422 interface. Therefore, APBUARTs 0 and 1 were selected for the system, communicating via the RS422 interface. These were connected to an RS422-to-USB converter, which ensured the serial data transmission between the board and the PC.

Finally, the injection of the software to be tested was carried out through the board's JTAG interface and orchestrated by the GRMON software, developed by Frontgrade Gaisler (2024) as a debug monitor for processors created by the company, including those in the LEON series. Although GRMON offers advanced features such as real-time reading and writing of system registers, only the download and application execution functionalities were utilized.

Emulated QEMU environment

As a counterpart to the real hardware mentioned in the previous section, an emulated machine was instantiated with resources analogous to those of the development board used. The emulated system features a single LEON3 core and fewer APBUARTs compared to the physical counterpart: two devices operating via TCP/IP interfaces. Despite the differences between the systems, these were not significant, as only one LEON3 core and two APBUARTs were required to execute the algorithm in both environments.

Despite the functional template provided by QEMU's original repository, it included only a single APBUART integrated into the system bus. Therefore, a second instance was manually implemented on the same bus using a different memory offset, but with the same device structure already coded by QEMU. It was confirmed that the second implemented component operated equivalently to the first, validating its functionality.

The injection of the software to be tested was performed during the execution and instantiation of the emulated machine, with the help of the MINGW tool.

Building upon the framework's structure and its key implemented components, the next section will provide a detailed analysis of the data flow and operations involved in conducting both unitary and sequential SIL tests, with a particular focus on the Caesar Cipher algorithm.

Software-in-the-loop (SIL) framework test flow

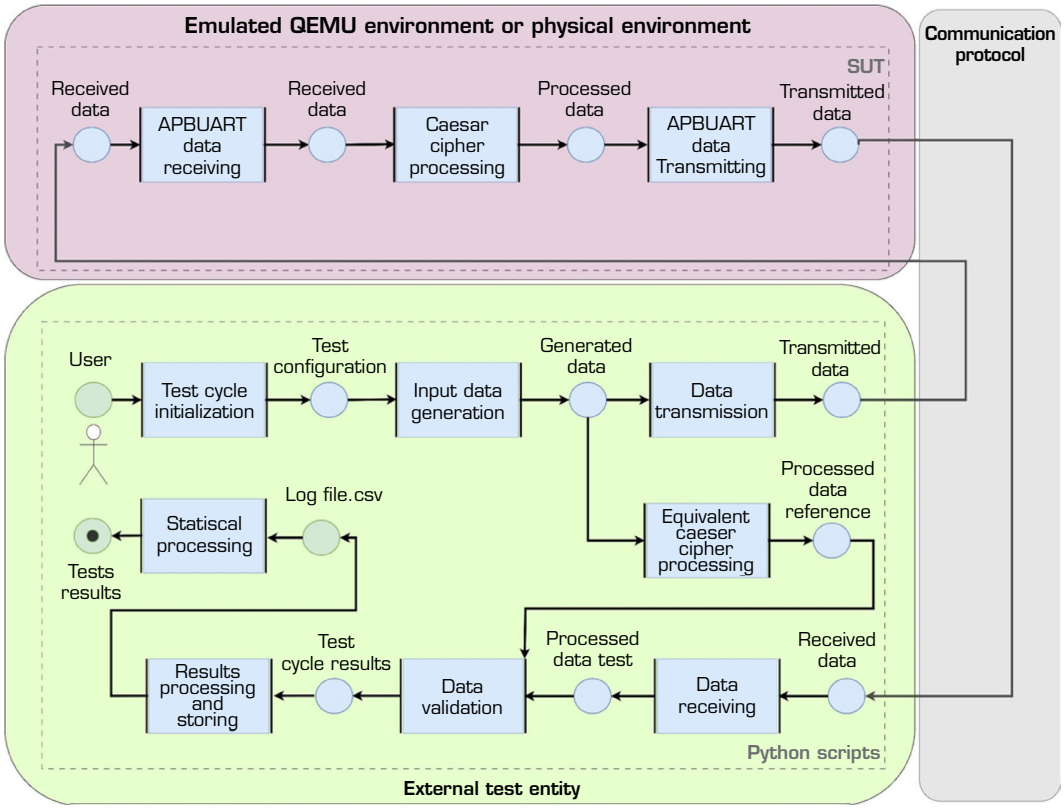
The primary component responsible for the data management and flow in the SIL framework is the test entity, as it controls the entire setup. However, before analyzing its operation, it is necessary to examine the SUT.

The SUT, represented in the red part of Fig. 3, was written in C and intended to run on both the emulated and physical environments. It initializes all the APBUARTs according to the desired configurations at a transmission/reception rate of 115.2 kBps. It then enters a loop to check the registers of each device: if the data receive buffer is empty, no action is taken; if the buffer contains data, it is received, converted into a string, transformed by the Caesar Cipher using an arbitrary offset and retransmitted through the same APBUART device. The loop continues to run until the system is manually shut down.

To show how the test entity complements the SUT in a test setup for both test environments, a Product Flow Schema (PFS) (Silva and Myiagi 1995) was created, which can be seen in Fig. 3.

The PFS in Fig. 3 represents the data flow of a unit test in the defined SIL framework proposed. The test flow is structured into eleven steps, as described:

1. Test cycle initialization: the test flow begins with the user interacting with the test manager through the terminal. During this step, basic test parameters are collected, including the number of unit tests, communication ports, and number of cycles. This information is then processed to generate the configuration parameters, which contain the data necessary for testing.
2. Input data generation: at this stage, an arbitrary and random set of data is generated based on the test parameters and on the format to be accepted by the SUT.
3. Data transmission: then, the external test entity sends the data set generated in the previous state to the APBUART selected for the current unit test. The data can be sent via RS422-USB or TCP/IP protocol, depending on the nature of the APBUART (real or emulated).



Source: Elaborated by the authors.

Figure 3. PFS of the architecture's unit test.

4. Equivalent Caesar Cipher processing: this step applies the Caesar Cipher algorithm internally at the test entity. It generates processed reference data, which will be used for comparison with the data processed by the SUT.

5. APBUART data receiving: in the QEMU emulated environment or the physical environment, the first step is to receive and prepare the data, making it available to the SUT being executed on the LEON3 processor.

6. Caesar Cipher processing: the data is then processed according to the Caesar Cipher algorithm by the SUT. After processing, the data is forwarded to the APBUART data transmitting phase.

7. APBUART data transmitting: after the algorithm execution by the SUT, the processed data is retransmitted to the test entity via RS422-USB or TCP/IP protocol.

8. Data receiving: in the external test entity, data packets are received and prepared for internal analysis.

9. Data validation: this step compares the data processed by the SUT with the original data processed in the equivalent Caesar Cipher processing for validation.

10. Results processing and storing: after validation, the results are processed and added into a .csv log file.

11. Statistical processing: finally, the log file is analyzed, producing temporal and logical statistical measures that summarize the overall results of the test conducted.

Based on the test cycle presented by the PFS, the testing framework is designed to integrate a series of unit tests into a comprehensive sequential test, each utilizing a specific APBUART. The execution loop iteratively cycles through the selected APBUARTs available, one at a time, until the predetermined number of unit tests is completed, thereby concluding the execution of the sequential SIL test. Upon completion, the framework processes the timing data recorded in a *.csv* log file, calculating the standard deviation and average timing for each APBUART used. The system then verifies the number of test cycles executed successfully for each device, assessing whether the algorithm performs correctly across both systems.

The structure of the SUT shows that each stimulus, data processing, and data reception are performed independently, including across different APBUARTs. This confirms the autonomous role of each device in every unit test or test cycle. Although each cycle and APBUART operates independently, it is important to note that the code is designed to retransmit data from previous iterations between the devices involved, resulting in data being shared among them. This configuration allows for simultaneous testing of the algorithm on both emulated and physical systems, and it also enables the evaluation of their interactions.

TESTS AND RESULTS

Based on the proposed and implemented approach seen in Fig. 2, three main SIL cases were established to validate the software on emulated systems compared to a physical system.

Initially, it was necessary to validate the software on real hardware to ensure the reliability of future tests carried out on virtualized systems. Then, the first test case integrated the external test entity to the GR712RC board. This test case, defined as the physical case, involved two sequential tests: one with a single real UART and another with two real UARTs. These tests were crucial for collecting baseline performance and success rate data, validating the test setup's reliability, and providing reference points for subsequent comparisons with emulated systems.

In the second case, it was important to perform the same tests on virtualized hardware using the implemented QEMU interface. The success of these tests and their compatibility with the results of the first case would indicate a high degree of fidelity between the virtual and real systems for performing SIL tests. This second case, named the emulated case, involved two sequential tests: one with a single emulated UART and the other with two emulated UARTs.

In the last case, dynamic tests were performed integrating the real and virtual systems, allowing SIL tests to be performed simultaneously on domains of different natures. This case, defined as the mixed case, involved two sequential tests: the first with an emulated UART and a real UART, and the second with two UARTs of each type. These tests analyzed the performance and success rates of each subsystem in an integrated environment. An alternative goal was to ensure that the coexistence of real and emulated components does not adversely affect their respective functionalities.

Three main test cases were proposed in total, each consisting of two sequential tests with different configurations, with one incorporating a predetermined number of unit tests. Table 3 helped to structure the objectives and requirements for each defined sequential test and to ensure a systematic analysis of the results derived from these tests.

The Test case column categorizes each test according to the previously defined criteria, distinguishing between those conducted on physical systems, emulated environments, or a combination of both. The Test condition column details the key aspects of each sequential test, with a particular emphasis on the characteristics of the UARTs involved. The Success criteria column lists

Table 3. Decision table for each SIL test to be done.

Test case	Test condition	Success criteria	Verification
Physical case	Test with one real UART	Correct processing by the Caesar Cipher algorithm, of all transmitted data	Check timing performance with other known cases
	Test with two real UARTs		Compare timing performance with the previous test
Emulated case	Test with one emulated UART		Verify timing performance and consistency of the emulated system compared to the previous cases
	Test with two emulated UARTs		Compare timing performance with previous case
Mixed case	Test with one real and one emulated UART		Compare each UART performance with their previous related cases
	Test with two real and two emulated UARTs		Compare timing performance with previous case

Source: Elaborated by the authors.

the parameters used to determine whether each test meets its objectives. Finally, the Verification column outlines the post-test procedures employed to assess the SIL system's performance, emphasizing the comparison between results from emulated and real-world scenarios to confirm the system's reliability and fidelity.

The table presented highlights that all tests are designed to verify and validate the test software, represented by the red section in Fig. 3, across various systems. Each test also evaluates the temporal performance and success rate of different system types and devices, facilitating the assessment and comparison of the SIL methodology in both real and emulated scenarios.

Throughout all test scenarios, the SIL framework setup, as depicted in Fig. 2, was preserved with only minor adaptations adjusted to the specific requirements of each test case. In each sequential test, the test entity executed 1,000 repetitions of unit tests, each involving 100 bytes of random data. In each unit test, the entity compared the received data against the expected processed output, thereby validating the execution. Upon completing the sequential tests, two key temporal metrics were computed for each UART device: the mean and standard deviation of the reception, processing, and transmission times for all unit tests. Additionally, the software validation success rate was evaluated for each execution, further ensuring the logical correctness of every test.

The following topics will present the results and metrics obtained for each sequential test conducted, according to the previously defined general test cases.

Physical case – Tests with the real hardware

The first test performed was the SIL test using only one UART of the real system (APBUART 0 peripheral). All executions were successfully validated, with the timing results presented in Table 4, and the timing difference between the executions remained consistent, exhibiting only minor fluctuations, as indicated by the standard deviation metric presented.

Table 4. Timing results for the test with one real UART.

UART	Standard deviation (ms)	Average value (ms)
Real APBUART 0	9.96	84.00

Source: Elaborated by the authors.

The second test was performed with two real UARTs (APBUART 0 and APBUART 1 peripherals of the GR712RC board). The SIL test executed data routing between the two interfaces, and all executions were validated successfully. The timing results can be seen in Table 5. It is interesting to note that the overall timing was very consistent with the timing of a single UART, as expected.

Table 5. Timing results for the test with two real UARTs.

UART	Standard deviation (ms)	Average value (ms)
Real APBUART 0	8.26	83.11
Real APBUART 1	7.54	82.93

Source: Elaborated by the authors.

Emulated case – Tests with the emulated hardware

With real data ready and reliable, the same tests were performed using the emulated QEMU system. The first test was the SIL test with a single virtual UART. All executions were successfully validated, with the timing results presented in Table 6. Notably, the standard deviation of the emulated UART was similar to the real one. Even more notably, the average time was lower than the real one.

Table 6. Timing results for the test with one emulated UART.

UART	Standard deviation (ms)	Average value (ms)
Emulated APBUART 0	8.50	64.26

Source: Elaborated by the authors.

The second SIL test was executed with two virtual UARTs, routing data to each other. All executions were successfully validated, with the timing results presented in Table 7. It is possible to see the same tendency of lower times as in the last test, when compared to the real UART. This timing difference can most likely be explained by the difference in speed between the physical UART RS422 interface and the TCP/IP interface. It would be expected that a physical serial interface would transmit data slower than a more advanced and faster TCP/IP one.

Table 7. Timing results for the test with two emulated UARTs.

UART	Standard deviation (ms)	Average value (ms)
Emulated APBUART 0	6.76	64.42
Emulated APBUART 1	7.86	63.79

Source: Elaborated by the authors.

Mixed case – Dynamic tests with emulated and real hardware

The previous test cases validated both the real and virtual LEON3 systems against each other from the SIL perspective, but a far more interesting use case would be the real and virtual systems being able to communicate with each other in the context of a test setup. The ability of a virtualized system to communicate with a real one in real-time allows for several useful application scenarios, such as software tests conducted in integrated environments that combine both physical and emulated components of a mission's satellite or space probe. Using the proposed Python-based test entity, it was possible to perform these types of mixed tests regarding UART communication.

The first SIL test mixed one real UART with one virtualized counterpart (both APBUART 0 peripherals) exchanging data between each other. All mixed executions were validated, and the timings can be seen in Table 8. It is possible to notice a very similar average time as the other tests, with the emulated UART being faster and more consistent than the real one.

Table 8. Timing results for the test with one real and one emulated UARTs.

UART	Standard deviation (ms)	Average value (ms)
Real APBUART 0	14.38	88.70
Emulated APBUART 0	7.05	64.23

Source: Elaborated by the authors.

The second and last SIL test of all cases is, arguably, the most interesting one. It mixes four UARTs in total, two real and two virtualized (using both APBUART 0 and APBUART 1 peripherals) exchanging data in real-time.

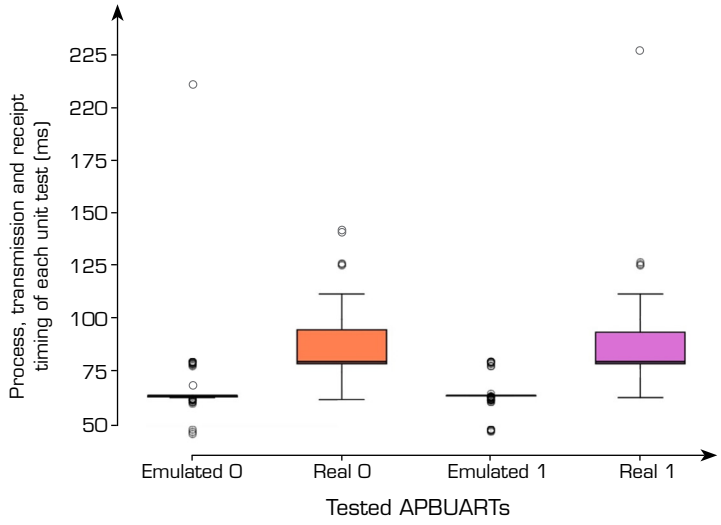
All executions were successfully validated, and the timings can be seen in Table 9. Again, the overall timing is very similar to the other tests, and the emulated UART appears to be faster than the real one.

Table 9. Timing results for the test with two real and two emulated UARTs.

UART	Standard deviation (ms)	Average value (ms)
Real APBUART 0	12.52	87.20
Real APBUART 1	14.10	86.46
Emulated APBUART 0	10.39	65.20
Emulated APBUART 1	5.40	64.58

Source: Elaborated by the authors.

To provide a clearer understanding of the timing data obtained from each device in the final sequential test, a boxplot chart, in Fig. 4, was created. This chart represents the average timing for each UART involved, represented by the groups "Emulated 0," "Emulated 1," "Real 0," and "Real 1." The central line within each box shows the median timing value, while the upper and lower edges of the box represent the first (Q1) and third (Q3) quartiles, respectively. This interquartile range (IQR) illustrates the



Source: Elaborated by the authors.

Figure 4. Boxplot chart representing the timing measures of each unit test regarding the last test sequential test done.

spread of the middle 50% of the data, and any points outside this range are considered outliers, shown as individual dots above or below the "whiskers."

By analyzing the boxplot, it is possible to observe that there is an average discrepancy of approximately 20 ms in the transmission, processing, and reception times between the different types of UARTs and the systems connected to them, consistent with the previous observations. Also, the emulated UARTs have a narrower IQR compared to the real UARTs, indicating that their timing measures are more consistent. This consistency suggests that the emulated UARTs achieve more precise timing results, with less variability in their transmission, processing, and reception times. In contrast, the real UARTs exhibit a broader IQR, highlighting greater variability in timing, which may suggest that real UARTs are more susceptible to timing fluctuations in this setup. Finally, it can be observed that, even without the need for further hypothesis testing, the analyzed systems exhibit distinct temporal characteristics, making it impossible to infer that data from the UARTs of different natures belong to the same probabilistic distribution and confirming the temporal discrepancies of both the emulated and physical systems and peripherals.

In an overall view, these data may indicate that the emulated system, in terms of temporal performance measurements, operates more consistently and slightly faster than the physical system under the same test cases, following the SIL methodology and the structured framework.

The results regarding all test cases carried out validate the proposed SIL testing framework and its architecture, particularly within an emulated environment applied to the LEON3 processor. They demonstrated that both systems operate correctly, although presenting a timing difference on average. Consequently, it is up to the framework's end-user to assess whether the timing variance between the two systems is significant for testing the algorithm in question.

Additionally, the tests facilitated real-time communication between virtual and physical systems in the same setup. This capability is crucial for enabling virtualized systems to effectively substitute physical ones in a software development testbed, for instance.

CONCLUSION

In the ever-evolving landscape of aerospace system development, the necessity for precision and reliable validation processes, allied with agile development methodologies, stands out as a major concern. More advanced computers and emulation software are enabling the virtualization of entire complex systems, which has the potential to considerably reduce development and testing time and cost.

Among several available emulation software, QEMU stands out as a widely used open-source alternative. Among its features, the ability to emulate the LEON3 processor is particularly interesting for the virtualization of aerospace systems. The default version of the QEMU's emulated LEON3 system is already powerful but can be expanded as needed to perform the emulation of more complex architectures.

An architecture for SIL testing was designed, utilizing QEMU-emulated machines along with hardware to execute the algorithms to be tested. For the architecture implementation, a virtual machine based on QEMU, including a LEON3 processor, was structured alongside a test entity and a physical system. The physical system included two LEON3 processor cores and other peripherals, based on the GR712RC board. Using UART interfaces, this setup allowed for the validation of algorithms according to various SIL testing implementation possibilities.

Based on the implemented proposal, the Caesar Cipher algorithm was validated according to the SIL methodology in three major test cases: a physical case, comprising only real hardware; an emulated case, comprehending only emulated hardware; and a mixed case, mixing emulated and physical hardware into the same test setup.

From a logical perspective, the algorithm was validated correctly in all three cases with total accuracy. However, an analysis of the timing metrics recorded in each sequential test within the three test cases revealed an average temporal discrepancy of 20 ms in the data reception, processing, and transmission times between the physical and emulated systems. Despite these differences, the mixed test case also demonstrated that systems of different natures can maintain continuous and valid logical interaction, with the test entity serving as an arbitrator in the presented setup.

It is evident that to effectively utilize the proposed approach in specific testing scenarios, certain precautions and considerations must be taken – especially given the notable timing discrepancies noted between its virtual behavior and that of its physical counterpart. For example, software tests in the framework that evaluate latency and performance during data transmission and processing – such as those involving the reception of TCs and the corresponding generation and transmission of TMs within a real-time system of a space mission – may not accurately reflect the behavior observed in a physical system. Conversely, tests aimed at verifying software sequentially and logically, such as those concerning the control of an embedded scientific instrument's electronic interfaces, can be conducted and interpreted with a higher degree of fidelity and accuracy.

Considering the proposed SIL testing setup and framework, it is important to note that, as a primary factor to be examined, the differences in communication protocols used for data transmission and reception between the physical and emulated systems may have contributed to the observed timing discrepancies, given that the serial protocol is notably slower than TCP/IP, assuming that data overload is not a concern. Additionally, it is possible to hypothesize that the behavior of the test entity might have also played a role in these differences, despite being structured symmetrically with respect to the reception, routing, and transmission of information among the connected systems.

For future research, it is recommended that a more detailed comparative temporal analysis be conducted between the presented emulated and physical systems within the framework, with the aim of enhancing the precision and fidelity of the emulated system relative to its physical counterpart. This analysis should focus on identifying and rectifying the observed timing discrepancies by examining differences in the communication protocols employed, as well as any potential interference from the test entity, and by proposing modifications to the framework's structure and architecture to optimize the overall timing accuracy of the emulated system.

Ultimately, if the timing differences between the systems are primarily attributable to their inherent natures, it is proposed that a study be undertaken to assess their implications across various test cases commonly employed in embedded systems within the aerospace sector. In this way, as briefly mentioned earlier, it will be possible to determine under which testing conditions these differences become significant and when they can be considered negligible, thereby facilitating the emulation of an environment with high accuracy and fidelity relative to one based on physical hardware.

The overall results presented are very interesting and encouraging. The work done is only an initial step towards the introduction of emulated systems as an alternative platform for testing and validating developing algorithms, particularly when applied according to the SIL methodology. Due to its efficiency, this resource would facilitate software development in

the aerospace sector, making it more compatible and symbiotic with the hardware being developed. In addition to improving the joint development of software and hardware for complex systems, this approach also drastically reduces production time and costs by strategically replacing intermediate testing and validation prototypes with high-fidelity emulated systems, which are easier to implement and use for testing. Thus, the expanded use of this technology indicates a potential paradigm shift in the validation of aerospace systems within this context, enabling the advancement of systems with progressively increasing complexity.

CONFLICT OF INTEREST

Nothing to declare.

AUTHORS' CONTRIBUTION

Conceptualization: França RM, Rofino F, and Santos LHA; **Formal Analysis:** Santos LHA; **Funding Acquisition:** Parro VC; **Investigation:** Rofino F and Santos LHA, França RM, Augusto SR, and Souza MAF; **Methodology:** Santos LHA, Rofino F, França RM, and Schneider CTT; **Project Administration:** Gueter DDV, França RM, and Parro VC; **Software:** Santos LHA, Rofino F, and Schneider CTT; **Visualization:** Santos LHA, Veiga JT, Rofino F, and França RM; **Writing – Original Draft Preparation:** França RM, Santos LHA, Veiga JT, and Rofino F; **Writing – Review & Editing:** Veiga JT, Santos LHA, França RM, Rofino F, Schneider CTT, Augusto SR, Souza MAF, and Parro VC; **Final approval:** Santos LHA.

DATA AVAILABILITY STATEMENT

All data sets were generated or analyzed in the current study.

FUNDING

Fundação de Amparo à Pesquisa do Estado de São Paulo
Grant No: 200857866-1

Conselho Nacional de Desenvolvimento Científico e Tecnológico
Grant No: 5740042008-4

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
Grant No: 170-15

ACKNOWLEDGEMENTS

The authors thank the Instituto Mauá de Tecnologia for supporting this development and Fundação de Amparo à Pesquisa do Estado de São Paulo, Conselho Nacional de Desenvolvimento Científico e Tecnológico, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CITAR Project and also Gisbert Peter of German Aerospace Center and his team for support with the project conception.

REFERENCES

Abudaqa A, Al-Kharoubi TM, Mudawar MF, Kobilika A (2018) Simulation of ARM and x86 microprocessors using in-order and out-of-order CPU models with Gem5 simulator. Paper presented 2018 5th International Conference on Electrical and Electronic Engineering. ADSRS Education and Research and Swinburne University of Technology; Istanbul, Turkey. https://doi.org/10.1109/ICEEE2.2018.8391354

Bekele YB, Limbrick DB, Kelly JC (2023) A survey of QEMU-based fault injection tools & techniques for emulating physical faults. IEEE Access 11:62662-62673. https://doi.org/10.1109/ACCESS.2023.3287503

Biagetti C, Falaschetti L, Crippa P, Alessandrini M, Turchetti C (2023) Open-source HW/SW co-simulation using QEMU and GHDL for VHDL-based SoC design. Electronics 12(18):3986. https://doi.org/10.3390/electronics12183986

Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S (2011) The gem5 simulator. ACM SIGARCH Comput Archit News 39(2):1-7. https://doi.org/10.1145/2024716.2024718

Buck ALL (1994) Ptolemy: a framework for simulating and prototyping heterogeneous systems. Los Angeles: University of California.

Charif A, Busnot G, Mameesh R, Sassolas T, Ventroux N (2019) Fast virtual prototyping for embedded computing systems design and exploration. Paper presented 2019 Rapid Simulation and Performance Evaluation: Methods and Tools. HiPEAC; Valencia, Spain. https://doi.org/10.1145/3300189.3300192

Choi JW, Nam BG (2012) Development of high performance space processor emulator based on QEMU – Open source dynamic translator. Paper presented 2012 12th International Conference on Control, Automation and Systems. IEEE; Jeju, South Korea. https://doi.org/10.1109/ISDFS58141.2023.10131693

Cordero F, Mendes J, Kuppusamy B, Dathe T, Irvine M, Williams A (2011) A cost-effective software development and validation environment and approach for Leon based satellite & payload subsystems. Paper presented 2011 5th International Conference on Recent Advances in Space Technologies. IEEE; Istanbul, Turkey. https://doi.org/10.1109/RAST.2011.5966889

Díaz E, Mateos R, Bueno EJ, Nieto R (2021) Enabling parallelized-QEMU for hardware/software co-simulation virtual platforms. Electronics 10(6):759. https://doi.org/10.3390/electronics10060759

[ESA] European Space Agency (2017) Plato – Revealing habitable worlds around solar-like stars (definition study report). Europe: ESA-SCI.

Ferrão RC, Augusto SR, Berni C, Santos FRFD, Parro VC, Plasson P, Gueguen L, Finco S, Peter G, Steller M (2016) Multi-purpose simulator for Plato mission: spacewire mission and applications, short paper. Paper presented 2016 International SpaceWire Conference. Osaka University; Yokohama, Japan. https://doi.org/10.1109/SpaceWire.2016.7771637

Ferrão RC, Augusto SR, da Silva TS, Parro VC (2012) Electronic simulator of the Plato satellite imaging system. J Aerosp Technol Manag 4:489-494. Available at https://jatm.com.br/jatm/article/view/218/262

Frontgrade Gaisler (2024) GRMON3 user's manual, version 3.3.12. [accessed 31 May 2024]. https://www.gaisler.com/doc/grmon3.pdf

Golkar A, Salado A (2020) Definition of New Space – Expert survey results and key technology trends. IEEE J Miniatur Air Space Syst 2(1):2-9. https://doi.org/10.1109/JMASS.2020.3045851

Gutierrez D, Ocampo W, Perez-Pons A, Upadhyay H, Joshi S (2023) Virtualization and validation of emulated STM-32 Blue Pill using the QEMU open-source framework. Paper presented 2023 11th International Symposium on Digital Forensics and Security. IEEE; Chattanooga, USA. https://doi.org/10.1109/ISDFS58141.2023.10131693

Habinc S, Glembo K, Gaisler J (2010) GR712RC-The Dual-Core LEON3FT System-on-Chip avionics solution. Paper presented 2010 Data Systems in Aerospace. Centre National d'Etudes Spatiales; Budapest, Hungary.

Hauschild F, Garb K, Auer L, Selmke B, Obermaier J (2021) Archie: A QEMU-based framework for architecture-independent evaluation of faults. Paper presented 2021 Workshop on Fault Detection and Tolerance in Cryptography. IEEE; Milan, Italy. https://doi.org/10.1109/FDTC53659.2021.00013

Illescas PB (2022) Smart hardware designs for probabilistically-analyzable processor architectures (master's thesis). Barcelona: Universitat Politècnica de Catalunya.

Kanavouras K, Hein AM, Sachidanand M (2022) Agile systems engineering for sub-CubeSat scale spacecraft. Paper presented 2022 73rd International Astronautical Congress; International Astronautical Federation; Paris. France. https://doi.org/10.48550/arXiv.2210.10653

Khurana R, Hodges S (2020) Beyond the prototype: understanding the challenge of scaling hardware device production. Paper presented 2020 Conference on Human Factors in Computing Systems. SIGCHI Executive Committee; Honolulu, USA. https://doi.org/10.1145/3313831.3376761

Kiesbye J, Messmann D, Preisinger M, Reina G, Nagy D, Schummer F, Mostad M, Kale T, Langer M (2019) Hardware-in-the-loop and software-in-the-loop testing of the MOVE-II CubeSat. Aerospace 6(12):130. https://doi.org/10.3390/aerospace6120130

Kopetz H, Steiner W (2022) Real-time systems: design principles for distributed embedded applications. London: Springer Nature.

Liechti C (2015) pySerial Documentation. Version 1.2.2. [accessed 20 February 2025]. https://pythonhosted.org/pyserial/#

Lonardi A, Pravadelli G (2014) On the co-simulation of SystemC with QEMU and OVP virtual platforms. Paper presented 2014 International Conference on Very Large Scale Integration-System on a Chip. IFIP/IEEE; Playa del Carmen, Mexico. https://doi.org/10.1007/978-3-319-25279-7_7

Marwedel P (2021) Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things. London: Springer Nature.

Mihalič F, Truntić M, Hren A (2022) Hardware-in-the-loop simulations: a historical overview of engineering challenges. Electronics 11(15):2462. https://doi.org/10.3390/electronics11152462

Mina J, Flores Z, López E, Pérez A, Calleja J-H (2016) Processor-in-the-loop and hardware-in-the-loop simulation of electric systems based on FPGA. Paper presented 2016 13th International Conference on Power Electronics. IEEE; Guanajuato, Mexico. https://doi.org/10.1109/CIEP.2016.7530751

Paikowsky D (2017) What is New Space? The changing ecosystem of global space activity. New Space 5(2):84-88. https://doi.org/10.1089/space.2016.0027

Peeters W (2021) Evolution of the space economy: government space to commercial space and New Space. Astropolitics 19(3):206-222. https://doi.org/10.1080/14777622.2021.1984001

Peeters W (2024) The paradigm shift of newspace: new business models and growth of the space economy. New Space 12(3). https://doi.org/10.1089/space.2023.0060

Probst M (2002) Dynamic binary translation. Paper presented 2002 Linux Developer's Conference. UKUUG; Bristol, UK.

Rodrigues GS, Kastensmidt FL, Bosio A (2022) Methodologies for testing and assessing electronic and computing systems. Cham: Springer. Approximate computing and its impact on accuracy, reliability and fault-tolerance. Synthesis lectures on engineering, science, and technology; p. 37-49. https://doi.org/10.1007/978-3-031-15717-2_4

Silva JR, Miyagi PE (1995) PFS/MFG: a high-level net for the modeling of discrete manufacturing systems. In: Camarinha-Matos L, Afsarmanesh H, editors. Balanced automation systems – BASYS 1995. Boston: Springer. p. 349-362. https://doi.org/10.1007/978-0-387-34910-7_33

SPARC International Inc. (1994) The SPARC architecture manual. Englewood Cliffs: Prentice Hall.

Sturesson F, Gaisler J, Ginosar R, Liran T (2011) Radiation characterization of a dual core LEON3-FT processor. Paper presented 2011 European Conference on Radiation and its Effects on Components and Systems. IEEE Nuclear and Plasma Sciences Society; Seville, Spain. https://doi.org/10.1109/RADECS.2011.6131334

Teich J (2012) Hardware/software codesign: the past, the present, and predicting the future. Proceedings of the IEEE 100(Special Centennial Issue):1411-1430. https://doi.org/10.1109/JPROC.2011.2182009

Weltzin C, Delgado S (2009) Using virtualization to reduce the cost of test. Paper presented 2009 IEEE International Automatic Testing Conference. IEEE; Anaheim, USA. https://doi.org/10.1109/AUTEST.2009.5314086

White J, Pilbeam A (2010) A survey of virtualization technologies with performance testing. arXiv1010.3233v1. https://doi.org/10.48550/arXiv.1010.3233

Ziemke C, Fritz DIM, Kossev DII, Brandt A, Eickhoff IJ, Roser HP (2011) An open-source system simulation framework for small satellite development based on opensimkit, qemu and systemc. Paper presented 2011 8th IAA Symposium on Small Satellites for Earth Observation; IAA; Berlin, Germany.