24th COBEM - 2017

24th ABCM International Congress of Mechanical Engineering
December 3-8, 2017, Curitiba, PR, Brazil

COBEM-2017-2888
HARDWARE AND SOFTWARE IN THE LOOP FOR V&V OF REAL-TIME EMBEDDED SOFTWARE IN AEROSPACE APPLICATIONS

Samoel Mirachi
Instituto Tecnológico de Aeronáutica – ITA
samoel@gmail.com

Emília Villani
Instituto Tecnológico de Aeronáutica – ITA
emilia@ita.br

Guilherme Correa
Instituto Tecnológico de Aeronáutica - ITA
stanice@gmail.com

Heitor Elisio Fernandes
Faculdade de Engenharia Elétrica e Computação – FEEC – UNICAMP
heitor.efernandes@gmail.com

Rodrigo de Souza
Embraer SA
rodrigotbw@hotmail.com

Gilson Maicon Maiolino
Orbital Engenharia LTDA
maicon@orbitalengenharia.com.br

Abstract. *This article proposes a framework for* ***anticipating verification and validation activities in the development of real-time embedded systems of space products****. The purpose is to detect implementation and integration errors in an early stage of the development lifecycle, in order to reduce costs and time delays. The framework, named Embedded System Incremental Verification (ESIV), combines the following environments: a Model-in-the-Loop (MIL), a Software-in-the-Loop (SIL), and two Hardware-in-the-Loop (HIL). The MIL environment runs in a single computer and is composed of a simulation model of the control law and the flight dynamics. Following, the SIL environment uses two desktop computers: one for the flight dynamics and another for running in real-time an early version of the on-board software. The first HIL includes a preliminary version of the on-board computer, a desktop to emulate sensors, actuators and the ground station, and second desktop for the flight dynamics. Finally, the second HIL uses the real sensors and actuators, the final version of the flight computer and the ground station equipment. The proposed framework is successfully applied to the on-board computer of the microgravity suborbital platform with the purpose of avoiding failures during the launch.*

Keywords: hardware in the loop, software in the loop, V&V, embedded systems, real-time systems, aerospace applications.

1. INTRODUCTION

This paper approaches the problem of anticipating V&V activities in the development cycle of real-time embedded software from aerospace domain. The main motivation is to reduce development time and costs through the early identification of design errors and HW/SW integration problems. Due to the critical nature of aerospace products, the development of embedded software follows waterfall model, as proposed by the standard *ECSS-E-ST-40C* (ECSS, 2009) and illustrated in Fig. 1.

Although the *ECSS-E-ST-40C* does have V&V activities since early phases of lifecycle, these activities usually concern the elaboration of a verification plan and tests specifications (Mirachi and Vaz, 2010). Real-time tests and HW/SW integration tests are usually performed when the hardware platform is available. As a consequence, some software implementation and integration problems are only detected after the Critical Design Review (CDR), when the engineering model of the embedded system is built (Barp and Vicentini, 2011).

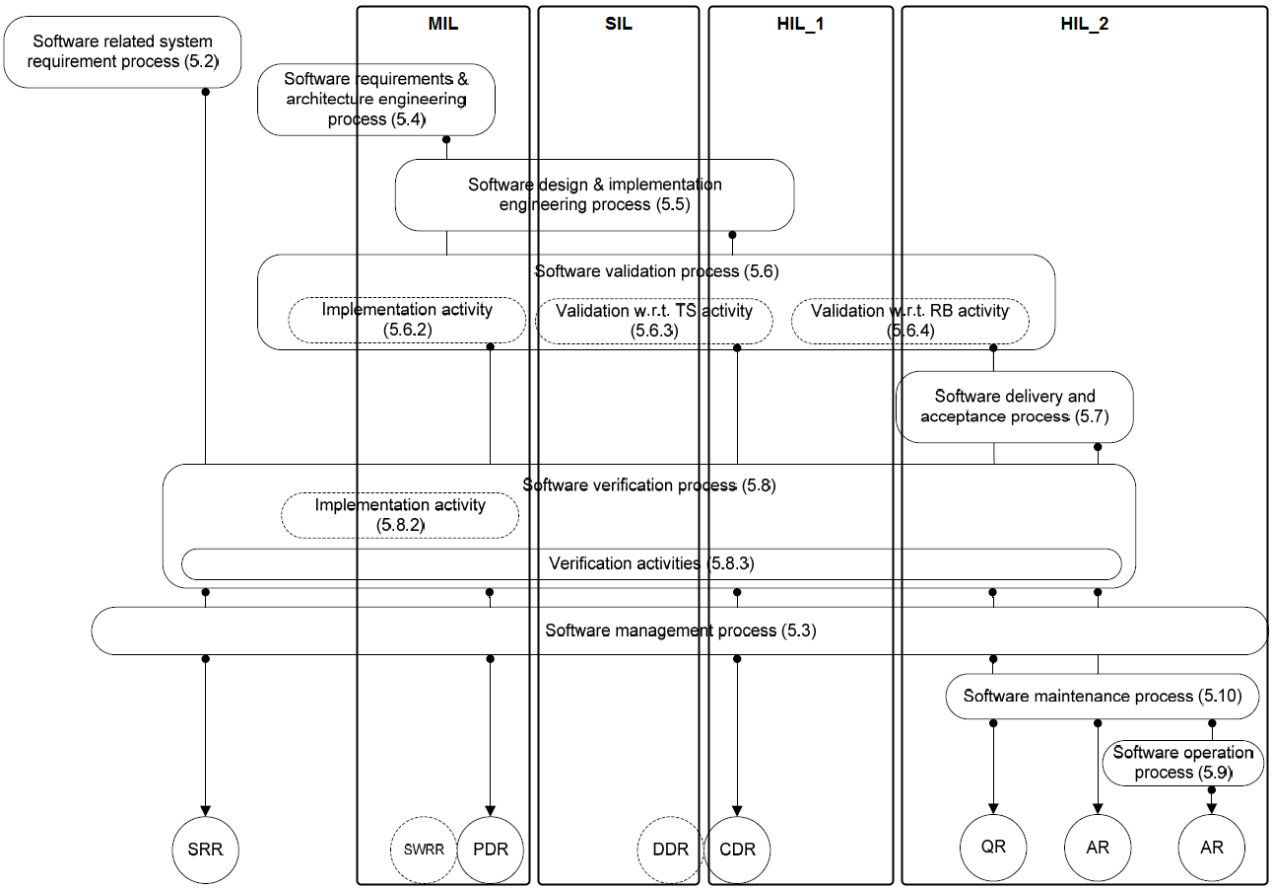

Fig. 1. Life Cycle Process (adapted from ECSS-E-ST-40C (ECSS, 2009))

In this context, this paper proposes a framework for the anticipation of embedded software testing in the aerospace domain (Laplante, 2004; Pressman, 2006; Sommerville, 2007). The Embedded System Incremental Verification (ESIV) framework combines software-in-the-loop (SIL) and hardware-in-the-loop (HIL) testing environments to provide the early identification of software errors and design problems. The distinct features of the framework are the real-time SIL environment and the building of two HIL environments. The SIL environment, built after model-in-the loop (MIL) testing activities, allows the execution of real-time tests without a specific HW platform. The first HIL is a low cost, modular environment that allows emulation of sensors, actuators, and its integration with a simplified version of the On Board Computer (OBC). The second HIL is high fidelity environment, where a complete version of the OBC is integrated to the actual sensors and actuators.

When comparing to other works in the literature, the main contribution of the ESIV framework is the introduction of simple and low-cost environments for detecting real-time and HW/SW integration problems in the early stages of the development phase, without the need of waiting for a high fidelity HW prototype.

The paper uses as a case study the OBC software of a microgravity suborbital platform, known by the Brazilian acronym PSM (*Plataforma Suborbital de Microgravidade*). The PSM is a project developed by *Orbital Engenharia*, a Brazilian aerospace company, with financial support from FINEP. It is a key project for the Brazilian aerospace sector and aims to nationalize the design and production of microgravity platforms. The motivation for the application of the proposed framework to the PSM is to avoid the faults/failures during launch, as occurred in the Cumã operation in December 2002, when the payload module had a premature separation from the VS-30 launcher (Mirachi and Vaz, 2011).

2. RELATED WORK

In this section, we discuss published works related to the use of in-the-loop practices in different engineering applications. Particular emphasis is given to works that combine MIL, SIL and/or HIL approaches for the development of critical embedded systems.

In the aerospace domain, (Wenboa and Qiangb, 2012) discuss the importance of using a HIL approach to validate with success the design of the navigation, guidance and control algorithms of a launch rocket. The physical devices used in the HIL simulation are the rocket borne computer and the steering gear. Differently from our proposal, the authors go directly from MIL environment to HIL environment.

Paw and Balas (2010) present a framework to support the progressive validation of flight control systems of UAVs. The framework combines the use of MIL, SIL and processor-in-the-loop (PIL) environments, before testing the controller in the final flying vehicle. One of the main contributions of the paper is the use of flight test data to improve robustness and performance of the model in use in all the three environments. Another work related to the development of UAV control system published by Mutter et al. (2011). In this case, the authors discuss the use of SIL for testing software implementation of UAV controller. A visualization framework based on Simulink VRML (Virtual Reality Modelling Language) is proposed to simulate the behavior of the UAV when submitted to a specific control routine. Among the advantages of the approach is the low cost of the solution and the possibility of integrating the controller software into the SIL environment without further modification. Comparing these works with our proposal, one immediate difference is that in the case of UAVs, the hardware platform usually is a COTS product and is available at the beginning of the software development. The main contribution of the MIL, SIL and HIL environments is to provide an incremental detection of errors associated with the incremental complexity of the testing environment.

In the automotive domain, Shokry and Hinchey (2009) provide an overview of the so-called X-IL approaches: MIL, SIL, PIL and HIL. They discuss the advantages of a progressive development, from MIL to HIL, as a way of tackling the increasing complexity of embedded software. The paper discusses how the X-IL environments are related to each other and the problems and errors each environment is usually able to identify. Although the paper focuses on examples from automotive industry, most of the conclusions are also applied to other domains. Another example from automotive industry is the work of Wegener and Kruse (2009). In this case, the authors advocate the use of evolutionary techniques for designing test cases, selecting and evaluating test data when performing functional tests in MIL, SIL and HIL environment. An automated solution is developed using commercial tools and applied to an ABS system in order to evaluate the evolutionary approach.

Muresan and Pitica (2012) also discuss the quality of testing activity. Their work evaluates the reliability of SIL environment using as an example the speed controller of a DC motor. Regarding accuracy of the results, SIL and HIL were equivalent. Regarding the capability of emulating a Real Time Operation System (RTOS), the authors observed that it depends on the performance of the host PC. One obvious advantage of SIL is the possibility of starting tests at the development phase, when the embedded system hardware is not available yet.

Kwon and Choi (1999) adapts the SIL principles for the verification of distributed control systems, using a distributed simulation environment composed of several personal computers (PCs) connected via an Ethernet network. The Distributed SIL (DSIL) framework is a low cost solution that allows the verification of real time requirements related to network communication. In a similar direction, Martinez et al. (2003) describes the implementation of SIL and HIL environments for communication networks and network-centric systems. In both environments, the part of the system that is represented by a model has a discrete event driven dynamics and is simulated by the OPNET modeler. The motivation of this work is the need of evaluating the performance of large and complex networks, combining virtual models with real hardware. Demers et al. (2007) also approach the problem of validating large communication networks using OPNET event-driven simulator. The authors point the main advantages of the SIL approach over HIL approach, such as low cost and flexibility to adapt the design solution. When compared to MIL, the main advantage is higher fidelity of the simulation. These conclusions are generic - they can also be applied to the aerospace domain and are confirmed in our work.

3. THE (EMBEDDED SYSTEM INCREMENTAL VERIFICATION) ESIV FRAMEWORK

The Embedded System Incremental Verification (ESIV) framework is composed of three distinct and incremental environments, each of which requires the previous one for its implementation.

Fig. 2 shows the framework block diagram, identifying the main processing units used at each stage. The starting point for building the framework is the Model-in-the-Loop (MIL) environment that includes a model of the control law and a model of the flight dynamics.

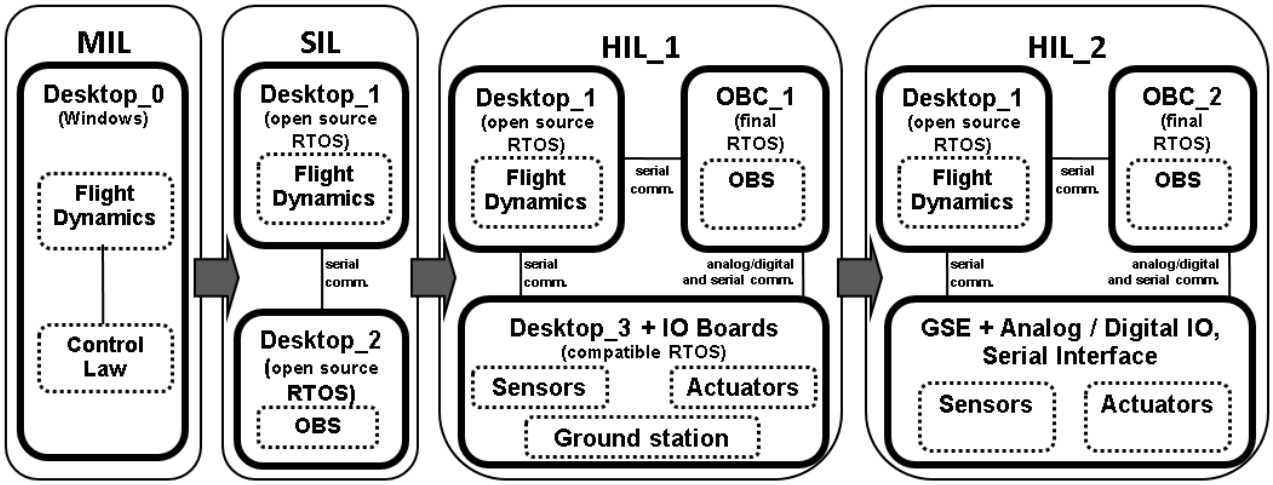

Fig. 2. Evolution of the proposed environment

3.1 Software-in-the-loop (SIL) environment

From the MIL environment, the SIL is built using two desktop computers running an open source RTOS and using a serial interface for communication. As depicted in Fig. 2, Desktop_1 runs the flight dynamic simulation. Desktop_2 executes an early version of OBS application, which includes the routines for flight control. It is developed based on the control law model of the MIL and it is coded, in order to guarantee the real time communication between these two applications.

The main purposes of the SIL environment are:

a) Once the flight control functions are coded into software routines and tested, errors introduced in the translation process can be detected in the SIL environment
b) Due to the RTOS used in the SIL environment, task priorities and scheduler configuration are introduced and verified;
c) By the fact the OBS is executed in real-time, it allows the verification of the time intervals for sending and receiving telemetry and telecommand data;
d) The simulation of the flight control routine is performed using inputs from flights of previous projects. These inputs contain disturbances not considered in the MIL environment and allow the test and verification of the control system robustness and limitations.

3.2 Hardware-in-the-loop environment 1 (HIL_1)

The first HIL environment is built from the SIL, maintaining the Desktop_1 with the fight dynamics simulation running in real-time. Desktop_2 is split into two different units: Desktop_3 and a commercial OBC, running the final RTOS that will be used in space. Desktop_3 simulates telemetry/remote data of sensors, actuators and the ground station receiver/transmitter. It is equipped with I/O boards and is able to communicate with the OBC using serial protocol, analog and digital signals. It must also run an RTOS compatible with the I/O boards used in the communication.

The OBC uses a hardware platform similar to the one that will flight, but not validated to space environment. It contains the final RTOS and code of the routines not only for flight control but also for data handling and housekeeping routines.

The main purposes of the HIL_1 environment are:

a) In early stages of development, HIL_1 allows the testing and verification of analog and digital I/Os of the OBC, without the need of having the real devices available (physical sensors and actuators). The development team can evaluate if a particular sensor with a specific calibration curve will meet the platform specification and if the sensor data reception and treatment routines are corrected;
b) HIL_1 supports real-time testing related to verification of interruptions, time-slice execution and pre-emptive scheduling. Communication delays can also be identified and corrected;
c) At a low cost, Desktop_3 plays the role of a ground support equipment (GSE) that receives and transmits (via serial communication) all telemetry/remote information to the OBC;
d) As the OBC contains the final OBS routines, coding and algorithm errors introduced in the translation from the SIL environment can also be detected and corrected.

3.3 Hardware-in-the-loop environment 2 (HIL_2)

Finally, HIL_2 is built from the HIL_1 by replacing the signals from Desktop_3 with signals coming directly from the real sensors, actuators and telemetry equipment. Also in HIL_2, the OBC is replaced by the final flight computer without the need of repeating all the OBC tests, once that the hardware boards (processors and memory) are similar and the code is exactly the same of HIL_1. I/Os connections are also the same. The differences between HIL_1 and HIL_2 are related to components, connectors, structure, and board cabinet.

The main purposes of the HIL_2 environment are:

a) It provides the confirmation of the tests performed in HIL_1, but now with the real sensors and actuators;
b) HIL_2 can be used to perform tests with the OBC installed in a gravity platform, in order to verify pitch, yaw and roll flight control routines;
c) Communication tests with final telemetry equipment are performed using HIL_2.

4. APPLICATION TO A MICROGRAVITY SUBORBITAL PLATFORM

This section describes the application of ESIV framework to the development of the embedded software of a microgravity suborbital platform.

In this example, the OBS is responsible to provide two main functionalities: (1) execution of the rate control routines, known as Rate Control System (RCS), and (2) on-board management of the platform and payload data, including telemetry/telecommand processing routines, commonly named Command and Data Handling (C&DH) (Wertz and Larson, 1999).

In order to provide a better understanding of the OBS behavior, Fig. 3 presents the operation modes (OM) of the OBS and describes how these two main functionalities are activated and combined during the platform operation. As shown in the diagram, when the OBC is turned on, the OBS enters in the "wait" mode, and waits for a confirmation to start the execution of an IF routine that will define the operational mode. The possible operational modes are: propulsion, uG, control, flat-spin and recovery modes. The IF routine is performed at each oscillator cycle and calls the RCS and/or the C&DH routines, according to the operational mode, until the end of the mission. At each mode, the RCS and C&DH runs in a different way:

In the propulsion mode, before lift-off, the RCS function is in idle and the C&DH exchanges telemetry and telecommand data via umbilical cable. After lift-off, the C&DH exchanges data via transmitter and receiver units.

In the uG mode, the RCS enters in operation in a simplified mode in order to maintain the system in microgravity, while the C&DH perform all its standard functions, including transmission of data from payload experiments.

In the control mode, the RCS operates in a different mode, including actuation at high and low pressure levels, while the C&DH operation remains as in the previous mode.

In the flat-spin mode, the RCS prepares the vehicle to system recovery, while the C&DH operation remains as in the previous mode.

In the recovery mode, the RCS returns to idle and the C&DH performs a different set of procedures, such as opening the parachutes and releasing color liquid that helps in its localization in the sea.

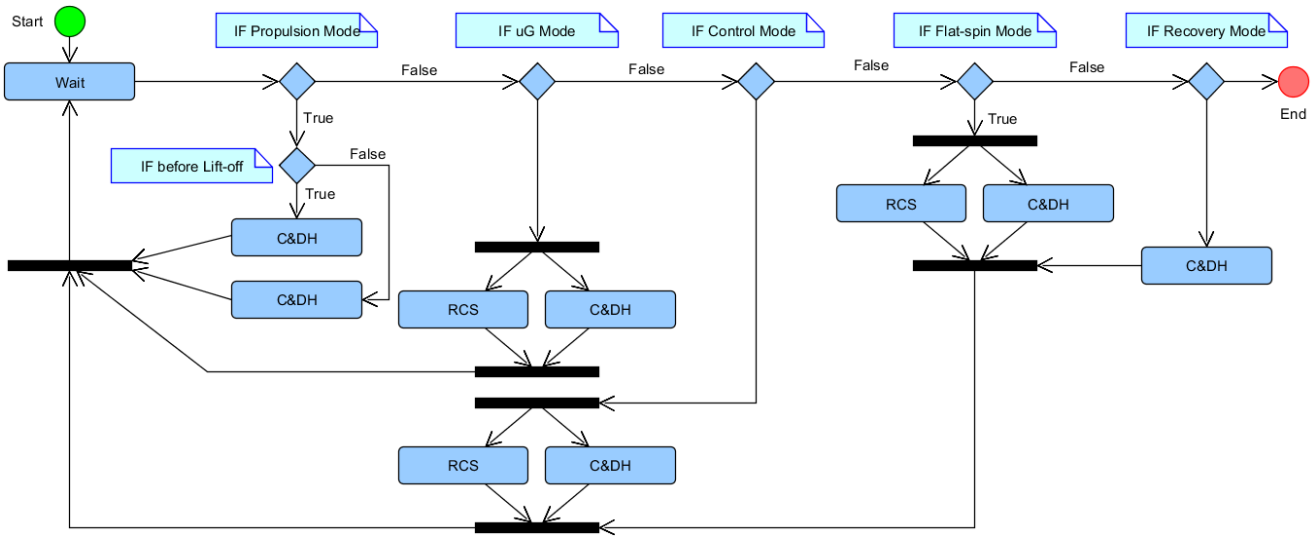

Fig. 3. State diagram of OBS operation modes

4.1 Overview of the implementation of ESIV framework

The list below brings some information about the implementation of SIL, HIL_1 and HIL_2 for the suborbital platform. The open RTOS used in both Desktop_1 and Desktop_2 is RTLinux, while the programming language adopted for both the flight dynamics and OBS is ANSI C. The solution adopted for Desktop_3 is based on COTS products from National Instruments and uses LabVIEW Real Time plug-in. The item description of ESIV framework there are the followings implementation:

- Open source RTOS (Desktop_1 and Desktop_2) →RTLinux;
- Flight dynamics programming language (Desktop_1) → ANSI C;
- Desktop2 programming language → ANSI C;
- Serial communication (SIL) → RS232;
- Desktop_3 I/O boards → NI boards;
- Desktop_3 RTOS → LabVIEW RT;
- OBC RTOS → VxWorks;
- Serial communication (HIL_1 and HIL_2) → RS422;
- Sensors (HIL_2) → Gyrometer, accelerometer, temperature and pressure sensors;
- Actuators → Ejector nozzles and cold gas pressure system.

OBC_1 and OBC_2 are based on similar hardware: both use the same processor and equivalent boards. However, while OBC_1 is equipped with plug and play I/O modules, OBC_2 has all the I/O integrated in a compact module. Furthermore, OBC_2 was modified in other to fulfil environment requirements of vibration, high temperature and radiation, among others.

4.2 Open RTOS implementation and verification

The choice of RTLinux as the open RTOS of Desktop_1 and Desktop_2 is based on the availability of on-line information and support, with the aim of minimizing the development time and maximize reliability of the system. RTLinux was proposed by Yodaiken 1996, consists of a RTOS microkernel that runs the entire Linux operating system as a fully preemptive process.

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|       -628|       -628|       1625|      10933|      10933|          0
RTD|        805|       -628|       1609|      13756|      13756|          0
RTD|       -781|       -781|       1622|      17424|      17424|          0
```

Fig. 4. RTAI latency test

For this work, the Linux kernel 2.6.23 RTAI from Debian GNU/Linux was compiled and installed. In order to verify the installation, latency tests are performed and an example of the corresponding results is illustrated in Fig. 4. It shows the minimum, maximum and average latency (lat min, lat max and lat avg) in each sample stage, as well as the overall values (ovl min and ovl max) since start up. The requirement of maximum latency for this application is of 50 ms, or 50.000.000 ns, which is satisfied by the current implementation of the RTLinux.

4.3 Implementation and verification of SIL software routines

For the implementation of the SIL environment, the software routines of Desktop_1 and Desktop_2 were developed using the Eclipse IDE.

In order to verify the rate control of the suborbital platform, the flight dynamics and RCS routines were manually coded in ANSI C, based on the MATLAB/Simulink models previously validated in the MIL environment. The flight dynamics receives as input from the RCS routine the current state of the propulsion actuators. It provides as output the angular speed in the 3 rotation axis (roll, pitch and yaw). The flight dynamics, implemented in Desktop_1, communicates with the Desktop_2 using an asynchronous serial port, with a transmission rate of 19200 Kbps.

Fig. 5 illustrates the package data exchanged between Desktop_1 and Desktop_2. Packet 0, 1 and 2 correspond to each of the 16 bits of roll, pitch and yaw gyrometer data, respectively. Each gyro data is composed of the real value of the angular velocity, given by the flight dynamics code and a white noise to simulate disturbances and sensor realistic behaviour. The line that begins with ":" contains the cold gas system actuation, each bit corresponds the positive or negative actuation, 0 for disable and 1 for enable.

```
packet0: 1111100001001001        packet0: 0111100100010000
packet1: 1000000000000000        packet1: 1000000000000001
packet2: 0111111111111111        packet2: 1000000000000001
:00000000:                       :00000001:
Timer: 3.550000                  Timer: 4.000000

packet0: 0001001000001010        packet0: 1000010111110000
packet1: 1000000000000000        packet1: 1000000000000001
packet2: 1000000000000000
:00000001:                        • • •
Timer: 3.700000
```

Fig. 5. Package data exchanged between Desktop_1 and Desktop_2

4.4 Implementation and verification of OBC_1 and OBC_2

In the HIL_1 and HIL_2 environments, the RTOS chosen for OBC_1 and OBC_2 is VxWorks, which has been designed for use in embedded systems with hard real-time requirements, deterministic performance and safety certification. The RTC and C&DH functionalities are organized in 4 main tasks: receive input data, on board control, on board management, and send output data. Each task has its own timeslice, controlled by the VxWorks scheduler.

The development of OBC_1 and OBC_2 software was also performed in the Eclipse IDE, using a GCC compiler customized to generate software for PowerPC processors. From SIL to HIL_1 environment, the RCS software routines are complemented with IRIG standard, sensors calibration curves and routines related to the integration with VxWork RTOS. Furthermore, the C&DH routines are added to the OBS. They are also coded in ANSI C and were developed based on the software specification and design artifacts.

The OBC specification states that it should run at 20Hz, i.e., with a real time cycle of 50ms. In order to meet the requirements of the ECSS-E-40 part 2B standard, the whole cycle must performed with a 20% margin, which means that the verification tests of OBC_1 must ensure that all four tasks are executed in less than 40ms and then it remains idle until the next cycle.

4.5 Implementation of Desktop_3

In the HIL_1 environment, Desktop_3 is responsible for simulation of sensors, actuators and ground station. In order to perform a rapid prototype development, the LabVIEW platform was chosen, which made possible the development of HIL_1 environment in less than 6 months. A backbone panel from National Instruments is used as interface board.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	SYNC	SYNC	SYNC	SYNC	SUB-ID	TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP	MODO_OP	AI	AI	AI	AI	AI	AI	AI	AI
2	AI	AI	AI	AI	AI_Temp	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI
3	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI
4	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	AI	STATUS_AI	STATUS_AI	STATUS_AI	STATUS_AI	DI	DI
5	DI	DI	DI	DI	DI	DI	STATUS_DI	STATUS_DI	STATUS_DI	STATUS_DI	STATUS_DI	STATUS_DI	STATUS_DI	STATUS_DI	DO	DO	DO	DO
6	DO	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
7	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
8	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
9	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
10	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
11	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
12	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
13	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
14	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
15	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS
16	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	GPS	EXPER	EXPER	EXPER	EXPER	EXPER
17	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
18	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
19	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
20	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
21	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
22	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
23	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
24	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
25	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
26	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER
27	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	EXPER	CHECKSUM	CHECKSUM									

Fig. 6 - Telemetry package in the IRIG-106 standard

In HIL_1, OBC_1 should send telemetry data to Desktop_3 via RS-422 interface, at 115200 Kbps baud rate. In order to test the telemetry packet, a parser was developed in LabVIEW and implemented in Desktop_3 following the IRIG 106 standard. The package frame is illustrated in Fig. 6. It is composed of 477 bits related to analog and digital sensors, GPS data, payload data, and additional information required by IRIG standard.

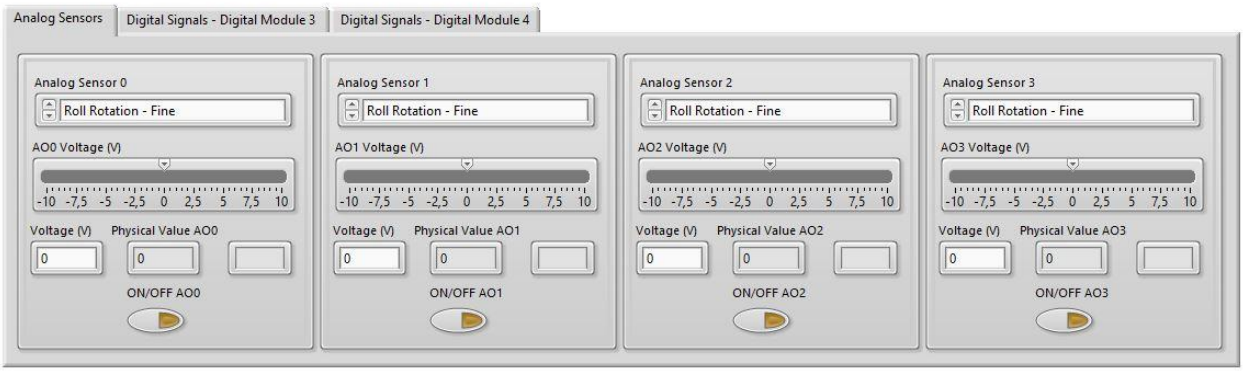

Fig. 7 – Graphical interface of analog sensors

Additionally, Desktop_3 has a set of routines for testing and verification of OBC input and output data. These routines can stimulate each input of the suborbital platform and check the corresponding output. Fig. 7 illustrates an example of the LabVIEW interface used to verify manually analog signals. This functionality supports the detection and correction of implementation problems, such as wiring and code errors, at early phase of project.

5. CONCLUSION

This paper proposes a framework for the incremental verification of embedded software of aerospace applications. The EISV framework combines model-in-the-loop (MIL), software-in-the-loop (SIL) and hardware-in-the-loop (HIL) environments allowing the early detection of errors and problems that otherwise would be detected only when a high fidelity hardware prototype is available, usually at the end of the CDR.

The contributions of the ESIV platform are assessed based on its application to the embedded software of a microgravity suborbital platform.

In the SIL environment, both the fight dynamics and the Rate Control System (RCS) routines run in two different computers that communicates using a serial interface. They are coded in ANSI C and executed in real-time. The SIL environment uses an open source operating system and tools, providing a low-cost environment to detect errors and problems related to the translation from the MIL model to the C code routine, the prioritization of tasks, and real-time execution. However, this system does not include the housekeeping and telemetry routines that compose the Command and Data handling (C&DH) system.

Following the SIL verification, a preliminary HIL (HIL_1) environment is proposed. It is composed of an early version of the OBC integrated with the flight dynamics and an additional computer equipped with I/O boards that simulates sensors, actuators and the communication with the ground station. At this point C&DH routines are integrated into the OBS. HIL_1 allows the verification of all OBC interfaces, testing of C&DH routines, and verifying tasks scheduling. In HIL1, it is possible to test all system I/Os and sensors wiring, as well as the calibration curve of all the sensors, without the need of having the real sensors and other hardware devices assembled.

The building of HIL_1 can postpone the need for the final HIL environment (HIL_2), which uses the qualifying model of the OBC and integrates it with the real sensors and actuators. It is important to observe that the SIL and HIL_1 aims at anticipating the detection of problems, so that when HIL_2 is tested, most of them have already been corrected.

6. ACKNOWLEDGEMENTS

This research has been supported by *Orbital Engenharia* LTDA. The Authors would like to thank FINEP for financial support.

7. REFERENCES

Alba, L., 2012. “Software in the Loop Environment Reliability for Testing Embedded Code”. In *IEEE 18th International Symposium for Design and Technology in Electronic Packaging – SIITME2012*.

Alba I., Romania, M.F., Gareis, S., Schätz, B., Bayha, A., Grüneis, F., Kanis, M. and Koss, D., 2011. "Model-Driven In-the-Loop Validation: Simulation-Based Testing of UAV Software Using Virtual Environments". In *18th IEEE International Conference and Workshops – ECBS2011*. Las Vegas, USA.

Barp, A. and Vicentini, M.B., 2011. *Aplicação de Tecnologia de Modelagem e Simulação de Sistemas no Desenvolvimento de Produtos na Embraer*. Embraer S.A., Brazil.

Demers, S., Gopalakrishnan, P. and Kant, L., 2007. "A Generic Solution to Software-in-the-Loop". In Military Communications Conference - MILCOM2007. Orlando, USA. DOI: 10.1109/MILCOM. 2007.4455268.

ECSS, 2009. *ECSS-E-ST-40C - Space Engineering – Software*. ESA Requirements and Standards Division, Noordwijk.

Kwon, W.H. and Choi, S.G., 1999. "Real-Time Distributed Software-In-the-Loop Simulation for Distributed Control Systems". In *IEEE International Symposium on Computer Aided Control System Design.* Kohala Coast, HI, USA.

Laplante, P.A., 2004. *Real-Time Systems Design and Analysis*. IEEE Press Editorial Board, Piscataway, 3ª Edition.

Martinez, R., Wu, W., McNeill, K., Deal, J., Haynes, T. and Bradford, D., 2003. "Hardware And Software-In-The-Loop Techniques Using The OPNET Modeling Tool For JTRS Developmental Testing". In *Military Communications Conference - MILCOM2003*. Boston, USA.

Mirachi, S. and Vaz, C.C., 2011. "PSM-PL-079-03 Plano de Desenvolvimento, Verificação e Validação dos Softwares Embarcados – PSM. Development Plan Document". Property of Orbital Engenharia S.A.. Brazil.

Mirachi, S. and Vaz, C.C., 2010. "PSM-ET-029-04 Especificação dos Softwares Embarcados – PSM. Technical Especification Document". Property of Orbital Engenharia S.A.. Brazil

Muresan, M. and Pitica, D., 2012. "Software in the Loop Environment Reliability for Testing Embedded Code". In *IEEE 18th International Symposium for Design and Technology in Electronic Packaging – SIITME2012.*

Paw, Y.C. and Balas, G.J., 2010. *Development and application of an integrated framework for small UAV flight control development*. Elsevier Ltd, USA.Pressman, R.S., 2006. *Software engineering*. Mcgraw Hill - Artmed. 752p.

Shokry, H. and Hinchey, M., 2009. "Model-Based Verification of Embedded Software". In *IEEE Computer Society – MC2009.* Ireland.Sommerville, I., 2007. *Software engineer. Person.* Prentice Hall, New Jersey. 8ª edition.

Yodaiken, M.B.V., 1996. "Real time Linux". 14 Set. 2017 <http://vyodaiken.com/wp-content/uploads/2015/11/real_time_1996.pdf.

Wegener, J. and Kruse, P.M., 2009. "Search-Based Testing with in-the-loop Systems". In *2nd International Symposium on Search Based Software Engineering.*

UK.Wenboa, H. and Qiangb, Z., 2011. "The Hardware-in-the-loop Simulation on the Control System of a Small Launch Vehicle". In *International Workshop on Information and Electronics Engineering (IWIEE).* China.

Wertz, J.R. and Larson, W.J., 1999. *Apace Mission Analysis and Design*. Space Technology Library, El Segundo. 3ª edition.

8. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.
The authors are the only responsible for the printed material included in this paper.