AN INTEGRATED SVF FOR REAL-TIME CLOSED-LOOP HIGH-COMPLEXITY SYSTEM SIMULATIONS

Juan Pérez[1], José A. Pulido[1], Pedro Palomo[1], Antonio Latorre[1], João S. Silva[2], Hugo D. Lopes[2] and Alberto García[3]

[1]Deimos Space, Ronda de Poniente 19, 28760 Tres Cantos, Spain. Phone/Fax (+34) 91 806 34 50/51.
[2]Deimos Engenharia, Av. D. João II, Lote 1.17.01 - 10º, 1998-023 Lisboa, Portugal. Phone (+351) 21893 3010.
[3]ESA-ESTEC, Keplerlaan 1,2201 AZ Noordwijk, the Netherlands. Phone (+31) 71 565 4886.
[1]{juan.perez, jose-antonio.pulido, pedro.palomo, antonio.latorre}@deimos-space.com
[2]{joao.silva, hugo.lopes}@deimos.com.pt
[3]Alberto.Garcia@esa.int

ABSTRACT

This paper presents an integrated Software Validation Facility where the application software target of the test runs on a flight processor representative emulator while its environment is executed on MATLAB/Simulink, being both parts executed as different processes on a single PC. In this way, SW developers have available a compact but complete facility, where they are able to accurately analyze and profile the SW in order to have a representative idea of its performance when running on real HW, while taking profit of having an easy access to tune and get valid data from the simulated resources, such as HW registers, sensor measurements or any other data available in the real system.

1. INTRODUCTION

Closed-loop testing is one of the most important means of testing SW components. Any closed-loop simulation is composed by a Software Under Test (SWUT) and a simulated environment which interfaces with it, taking the signals output from the simulation environment and using them as input for the SWUT. Then, the SWUT performs its job, generating a set of outputs. In contrast to open-loop testing, where the strategy is restricted to the analysis of outputs when excited by controlled inputs, in closed-loop testing the set of outputs generated by the SWUT is then fed back into the environment simulation, thereby affecting it under actual service conditions and modifying the subsequent inputs to the SWUT.

Closed loop testing not only checks the behaviour of the particular SWUT, but also tests the reaction of the full system to the SWUT operation. This provides for a realistic platform for testing the environment, the SWUT and the interaction between both of them.

This type of testing simulates most closely the actual performance of the whole system. However, the requirements for this kind of testing platforms may also be complex, as it can require the exchange of a large amount of data and signals and the environment must also mimic the real physical system and provide the full response during closed-loop testing. Depending on the type of system under test, the expected fidelity of the simulation and the sensitivity of the SWUT to the fine grain parameters of this system, the environment simulation may require to perfectly emulate complex physical or behavioural processes, which are often difficult to model without the use of advanced simulation tools such as MATLAB/Simulink®. Moreover, a Software Validation Facility (SVF) for real-time closed-loop simulations must offer to the SWUT simulated real-time performance, together with synchronization means to ensure a fully correct timing relation between the SWUT and its environment.

This paper presents a SVF where both, the SWUT and its environment are executed as different applications in the same PC that communicate through a shared memory interface, allowing fast data transfers and the use of synchronization elements. The synchronization between the environment simulation and the SWUT and the simulated time coherence between them (tackled in section 2.4) are achieved implementing protocols that establish the order of execution and data flow between the elements and adjust the execution times of the "real-time friendly" processor emulator to the less flexible time reference of the environment simulation, implemented in Simulink.

The SVF described in this paper has been successfully used in the development of the GNSS Dynamics Simulator and AGGA-4 Test and Simulation Tool (GSTST) project, where the main objective was to develop a tool to design, analyse and test GNSS algorithms for AGGA-4-based GNSS receivers[1][2].

2. INTEGRATED SVF ARCHITECTURE

Simulink has become in one of the most relevant environments for multidomain simulation and Model-Based Design. It is integrated with MATLAB, enabling you to incorporate algorithms into models and export simulation results for further analysis.

Given the penetration of this development environment in the engineering industry, nowadays it is probably the main alternative for development of closed-loop simulations, involving the modelling of the environment in Simulink and its translation into code through an autocoding process (the code is then executed for instance in dedicated platforms such as dSpace®). However, sometimes the autocoding process is not

feasible or convenient (e.g. if frequent model modifications are necessary). It is on these cases, and especially for the simulation of hybrid HW/SW systems, where the present paper is focused, as our purpose is to build a SVF where the environment is executed directly in Simulink and not translated to code.

Regarding the SWUT, space processor market for ESA missions is currently dominated by the LEON family, and these are also the processors used in ESA's AGGA product line (in which this SVF is firstly used), so the SVF has been designed with this idea on mind. The architecture and tools employed should be used straight forward with the LEON2-FT processor or equivalent emulator and easily extended to other LEON processors.

2.1 Design Alternatives

With these premises, two possible approaches for the SVF architecture were analysed (Fig. 1):

A. SWUT running on a LEON2-FT processor (requiring the development of an interface to support the communication between the Simulink model and the HW);
B. SWUT running on a LEON2 processor emulator (on the same PC as the Simulink model, precluding the need for the implementation of a complex interface with the HW emulation and without synchronization limitations).

In option A, the interface to be developed in order to ensure the communication between Simulink and the LEON processor must have a physical layer and a SW layer (part of which would be included in the Simulink part while the rest would run on the LEON processor). Measurements sent to the SWUT and feedback signals generated by it are exchanged through this interface.

In option B, the SWUT runs on a LEON-2 emulator (e.g. TSIM [5]), using a software-only interface. Communication between the Environment and the SWUT would be assured through shared memory.

2.2 Trade-off and Selection

Both options have advantages and disadvantages, as enumerated below. The main advantage of option A is representativeness, as the SWUT runs directly on the LEON processor, which is the final target environment. Nevertheless, it is also worth noting some drawback about portability, besides the need to have an external board (the Development Board) connected to the Simulation PC, which makes this solution less portable.

On the other hand, the advantages of Option B (SWUT running on a LEON processor emulator) are the following:

- **Portability**: It would result in a software only solution, which means that additional hardware is not required and that the portability of the SVF is greatly increased;
- **Flexibility**: It offers a high level of flexibility, allowing the use of user-defined I/O interfaces, the possibility of linking the emulator with other user applications and the capability of tuning the emulator characteristics as needed.

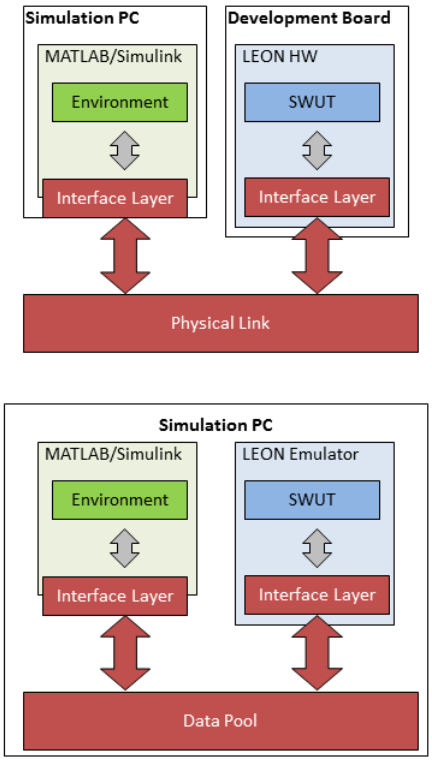

Figure 1: SVF design alternatives

However, this approach also has some disadvantages, namely:

- **Realism**: The use on an emulator instead of the real target has some limitations in terms of simulation representativeness, as some features may not be fully realistic. In the case of TSIM, the accuracy and generation of IEEE exceptions is host dependent and not always identical to the actual ERC32/LEON hardware. Still, the average timing accuracy of integer instructions is better than 0.5% and floating-point instructions have a typical accuracy of 2%. Nevertheless, these limitations may not be relevant for most of the software development phases, as it is still possible to test the algorithm performance and to estimate rather accurately the real-time behaviour of the development;
- **Speed**: Depending on the host PC and on the selected LEON's clock frequency, the simulation rate of the emulator may be slower than real-time. However, since an important part of the impact on simulation speed is expected to be on the Simulink side of the SVF, this is not expected to be an important limitation.

An important issue regarding option B is the selection of the LEON emulator tool, which has been also identified as one of the main cost drivers. Three alternatives have been considered: TSIM (the most popular LEON emulator, but expensive), the Terma Emulator Suite [3] and the QERx emulator [4].

A trade-off has been made and it was concluded that the TSIM alternatives, although eventually eligible in the future, were not guaranteed to be a timely option to

Deimos due to validation and legal issues affecting the Terma Emulator Suite and QERx respectively, so the TSIM emulator was selected for use in the SVF.
The final cost of options A and B may be different for each company depending on existing and reusable elements. Assuming a project starting from scratch, where none of the required software and hardware is available to a potential user, both options are similar because the difference between the cost of TSIM (option B) and a development board plus the GRMON software (option A) is almost negligible.
Table 1 summarizes the trade-off between both options.

Table 1: Trade-off between design alternatives

	LEON Board (Option A)	**LEON Emulator (Option B)**
Complexity	HIGH	HIGH
Portability	MEDIUM	HIGH
Flexibility	LOW	HIGH
Realism	HIGH	MEDIUM/HIGH

Therefore, taking into account the advantages and disadvantages presented above, the selected approach is option B, which also seems to be the most interesting for ESA (in the context in which the tool was developed and its adoption for future projects and missions) and for other potential users of such a tool.

2.3 Environment-SWUT Data Interface

The interface between the SWUT and the environment is designed with three main objectives, namely, to:

- Establish a synchronization scheme for the flow of information and processing, in order to obtain a time-true emulation.
- Allow the communication between both sub-systems, defining an accurate exchange of information using the available low-level features of the host operating system.
- Minimize the modifications to the SWUT required to port it to a Hardware-in-the-loop or real platform, making the interface with the simulated environment as representative as possible.

The layered architecture of the interface is displayed on Fig.2. On the simulated environment side, a MATLAB/Simulink environment holds the top level system, which runs each module at the required frequency. On the other side, a SWUT Test-Bed supports the execution of the user-developed algorithms (the SWUT) and implements also an interface, in charge of enabling the communication between both modules and providing an abstraction layer between them.
The connection with the SWUT Test-Bed interface is implemented in Simulink by using one or more S-Function blocks developed in C language with access to the underlying host operating system (for using disk files or communication devices). The inputs and outputs of the S-Function blocks are implemented as MATLAB Bus objects, which explicitly define each signal type and size. These Bus objects translate directly into C structures and therefore allow the definition of a single interface for the SWUT Data Interface.

On the SWUT Test-Bed side, the application is run on top of an embedded operating system designed for the LEON2 processor (RTEMS) and uses the TSIM I/O modules to access the host operating system features.
A shared memory object is employed to establish communication between both sides. Due to the modular design of the interface, a common static library has been implemented to provide an abstraction of the host operating system layer and allowing the portability of the platform. The interface of the static library is composed of simple operations to read/write the shared memory and to send/receive signals for synchronization.
The process of sending data from the environment to the SWUT Test-Bed to be used internally is as follows:

- The data obtained in the Environment is directed to the S-Function that implements the interface with the SWUT Test-Bed.
- The interface formats and packages the data to be sent, and delivers it using the host operating system facilities. After the information has been sent, it waits until the reception of a confirmation signal from the SWUT Test-Bed.
- The SWUT Test-Bed receives the data and delivers it to the SWUT algorithms for its processing. When the process is complete, a signal is sent to the Environment to finish this processing cycle.

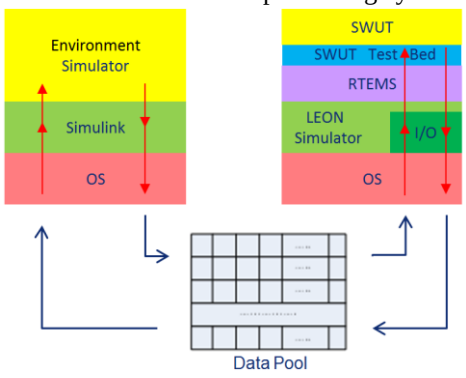

Figure 2: Environment-SWUT data interface

2.4 Environment-SWUT Time Synchronization

Temporal synchronization between the Environment Simulator and the SWUT is required, because they are run as independent processes using different references.
The Environment Simulator runs on Simulink (no real-time at all, just simulation steps), resulting in executions that could be longer or shorter than the period defined for the different system tasks.
The SWUT is run on a simulated real-time platform (with a real-time operating system and simulated HW, including the clocks and the operation count), which in turn, is running in the context of a Host Operating System (Windows). In other words, while the SWUT is run on simulated real-time (thanks to TSIM features), the user may experiment inconsistent execution global performance depending on several factors such as the simulator performance, the Host PC HW or the number of external processes running on Windows which interrupt the execution of the simulator.
Moreover, some other characteristics of the system must be taken into account:

- The Environment Simulator may have different MATLAB/Simulink modules running at different frequencies, each of them generating data that will be made available to the SWUT.
- The Environment Simulator may receive data from the SWUT at different frequencies to be used by different modules.
- Neither SWUT frequencies nor sequence of operation are known in advance. The SWUT shall be able to be configured to receive data at different frequencies.

In our implementation, the SWUT Test-Bed is in charge of the synchronization with the Environment Simulator. Two different patterns can be developed:

- Full Synchronized Pattern: the SWUT is executed when a synchronization signal is received from the Environment Simulator, after finalization of a simulation step. While the SWUT is executed, the Environment Simulator is blocked until the SWUT execution minimum period has finished and vice versa. There is no concurrence between the two sides of the simulation. This pattern is appropriate when both sides need the outputs of the other side.
- Semi-Synchronized Pattern: the SWUT is composed of one or more tasks that execute with different frequencies, in parallel to the execution of the Environment Simulator. The mechanisms to synchronize the execution of the SWUT with the Environment simulator are the messages from the Environment to the SWUT. When the Environment simulator detects that the last message sent has been taken by the SWUT, it continues its execution without waiting for the execution of the SWUT. This strategy allows parallelizing the execution of the both sides, so it is appropriate when Environment simulator does not need the outputs produced by the SWUT.

In order to perform closed-loop simulations, we have implemented the Full Synchronized Pattern, with the time master in the LEON Simulator side. Therefore, the Environment simulator executes a step when the synchronization signal is received from the SWUT, which remains blocked until new inputs are ready.

2.5 Execution Model

A first approach to the definition of the architecture was made by defining a periodic task in the SWUT Test-Bed that would run at the same frequency as the base step time of the GNSS Simulator, but this was later discarded for the following reasons:

- The synchronization task in the SWUT Test-Bed has additional duties, as the processing of incoming data from the GNSS Simulator to translate it into the desired interface to be provided to the SWUT, and the generation of monitoring data. These operations may require a non-negligible processing time, which should not be deducted from the CPU time available for the SWUT execution.
- Isolating the execution time of the SWUT allows realistic performance measurements, and can be used to define a duty cycle according to the time budget allocated to the SWUT in the final system.

The implemented architecture, which solves the previous issues, can be summarized as follows:

- A Timer is implemented in the SWUT Test-Bed with the same frequency as the highest frequency task in the whole system (e.g. if there is a module in the Environment simulator running at 1KHz, the Timer shall have a duration of 1ms) .
- A Synchronization Task is implemented, that is enabled when the Timer ends and interrupts the execution of the SWUT, entering in Simulator Mode using the IO modules functionality (in Simulator Mode, the simulator pauses all the HW). In this mode a SW Bus analyses the messages ready to be sent to the Environment Simulator.
- The mechanism to send messages from side to side is a memory mapped file, where each message has space reserved and a status word which controls if the message has been already read or not.
- In the Environment simulator side, the model waits for new messages. When it is notified to run a new cycle, the model reads the new messages, executes the simulation step and writes all output messages in the shared memory mapped file. Afterwards, the Environment simulator waits to be signalled again to execute the next simulation step.
- The SW Bus, in the LEON emulator side, analyses the received messages and puts the data in the positions to be read by the SWUT. The SW Bus returns the control to Synchronization Task, which arms the Timer again and goes to sleep, allowing the SWUT to continue its execution.

Fig.3 shows a chronogram with an excerpt of the execution of an application composed by:

- A GNSS Simulator (the Environment) working at a maximum rate of 1 KHz.
- The SWUT with two application tasks, Task A at 1 KHz and Task B at 500 Hz.

2.6 Results Processing

The processing of outputs from the SWUT and other monitoring data is performed on the MATLAB/ Simulink environment. The S-Functions that implement the communication with the SWUT provide as their outputs the SWUT output data as bus objects, which include the required feedback outputs to perform a closed loop simulation and to allow the comparison of the SWUT outputs to reference data generated in the Environment Simulator. Fig.4 shows an example of SWUT output (velocity computed on a GNSS receiver by the SWUT), compared with equivalent reference data.

The monitoring data generated by the SWUT Test-Bed is also available in the MATLAB environment. This information allows the analysis of the performance (fidelity, real time behaviour, errors) of the SWUT.

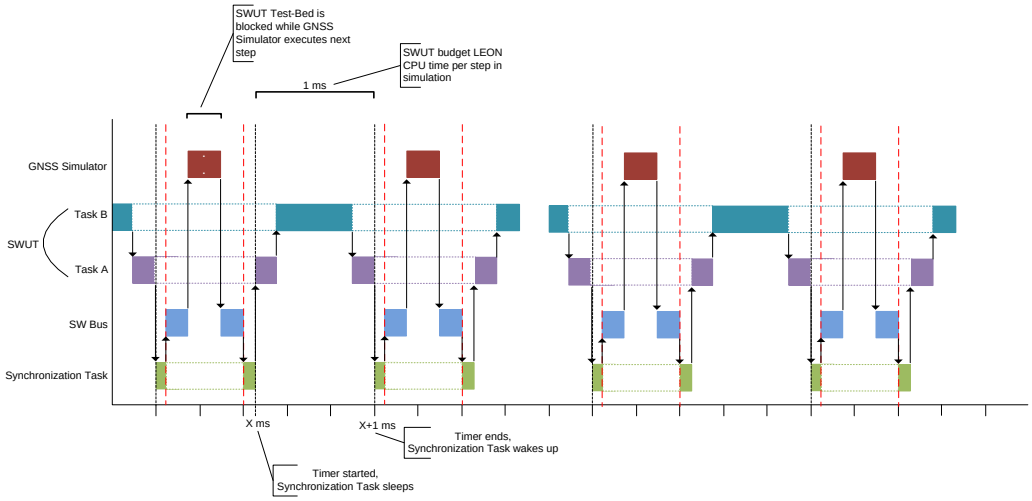

Figure 3: Time synchronization and data exchange

3. APPLICATION TO GNSS RECEIVERS

The SVF described in this paper has been successfully used in the GNSS Dynamics Simulator and AGGA-4 Test and Simulation Tool (GSTST) project [1], where the main objective was to develop a tool to design, perform a realistic analysis and finally, test GNSS signal processing and navigation algorithms for AGGA-4-based GNSS receivers. AGGA-4 is the Advanced GPS / Galileo ASIC developed by ESA for GNSS space applications [2]. The GSTST tool provides a relatively inexpensive solution (when compared to currently available hardware-based solutions) for the simulation of realistic GNSS observables and measurements (as the software part of the receivers would see them) as well as of the AGGA-4 programming registers, allowing AGGA-4-targeted software to be tested without the need for the AGGA-4 chip or complex and expensive hardware setups. GNSS signal emulation is performed in MATLAB/Simulink by the use of DEIMOS' GRANADA GNSS Blockset (formely GRANADA Factored Correlator Model Blockset) [6].

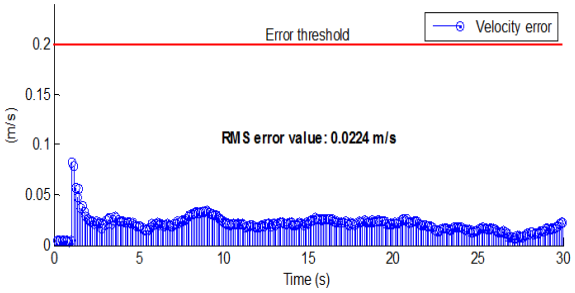

Figure 4: Example of SWUT output processing

The data exchanged between the GNSS Simulator and the SWUT Test-Bed can be sub-divided into the following categories:

- Hardware registers, which include 14 AGGA-4 programming registers, IE and ME observables (20 elements) per AGGA-4 channel (36 channels) and interrupt signals.
- Shared GSTST variables, which include GNSS measurements (more than 10 post-processed observables), ephemeris, status flags and other GSTST internal variables and outputs (including user-definable outputs) per channel.
- Shared SWUT variables, which include SWUT internal variables and outputs (including user-definable outputs), configuration parameters, control signals, and performance monitoring signals (for SWUT profiling) per channel.

Fig.5 shows an schema of the SVF configuration, where the different modules of the AGGA-4 are simulated on the Simulink side and communicate with the SWUT where the Tracking Loops algorithm and/or the Navigation Filter are executed on top of the LEON-2 Processor Emulator.

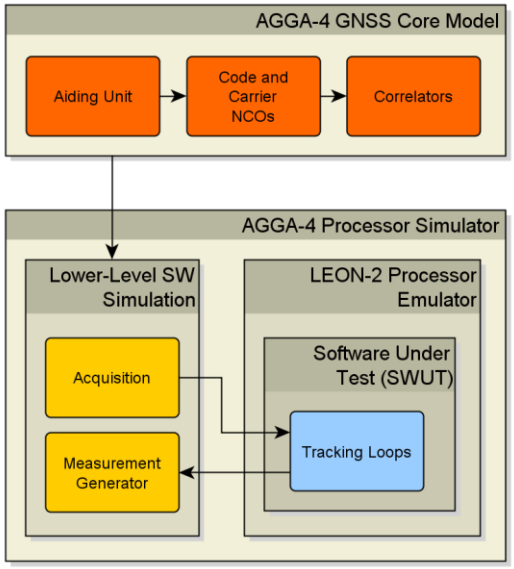

Figure 5: Example of SVF Usage for GNSS algorithms

In the scope of this project, two SWUTs have been developed to demonstrate the usage of the SVF in different configurations:

- Navigation Orbital Filter, with low frequency requirements and no parallelization. This SWUT implements the API to handle the Measurement Epoch signal generated in the GNSS Simulator with a frequency of 10 Hz. The SWUT receives measurement signals as well as satellite navigation

data, and is capable of PVT generation in LEO and GEO orbits, as well as computation of Doppler aiding signals to be fed back to the simulated tracking loops.

- Tracking Loop algorithms, with high frequency requirements and multiple channels processing. This SWUT has to attend different signals (Integration Epoch and Long Epoch) for several processing channels, which can be generated at a frequency of up to 1 Khz. This is done by implementing the APIs to handle both types of signals. In this case the SWUT obtains the correlator outputs and controls them by generating the required feedback signals.

Thanks to the flexibility of the SVF the performance of the SWUT can be analysed and refined. For example, the Navigation Orbital Filter implemented has been tested with different configurations in the Environment Simulator describing different scenarios of satellite constellations. Fig.6 displays the different performance of the algorithm achieved in a scenario with GNSS satellite outage.

The performance of the filter has also been analysed regarding its execution time and by tweaking the simulated system clock on the TSIM emulator, obtaining the results displayed in Fig.7.

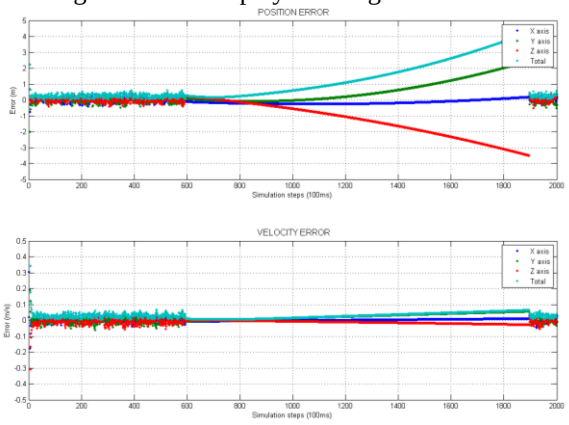

Figure 6: Performance Analysis – NOF application

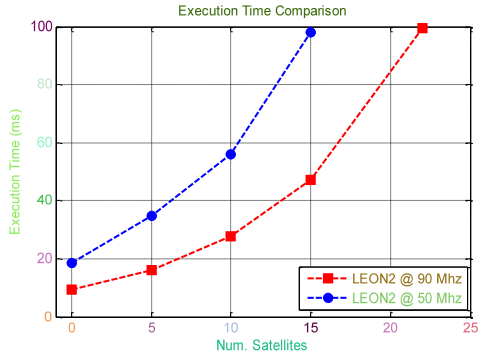

Figure 7: Execution Time Analysis – NOF application

4. CONCLUSIONS

A Software Validation Facility has been presented to support the development, analysis and test of a complex hybrid system where the SW algorithms interact with other parts of the system implemented in specific HW, simulated in Simulink.

This tool provides a low-cost solution for a system developer to execute closed-loop simulations in a layered and realistic environment (including real-time constraints and real reaction of the system to the SW outputs), enabling the possibility of tackling an iterative and incremental development process, with frequent modifications of the SWUT and the environment, including their boundaries (i.e. a SWUT composed by N modules can be developed in an incremental approach implementing and validating module by module, simulating the other ones in Simulink, so what belongs to the SWUT and to the Environment simulator is different for every development stage).

This way, the SVF allows to perform a wide range of tests (performance, accuracy, etc.) from early stages of the development process without the need for complex and costly HW setups.

The presented SVF has been successfully used in the GSTST project, designed to support the validation of navigation algorithms candidate to be used with real AGGA-4-based receivers in future missions (e.g. GRAS-MetOp 2 [7] and other GNSS receivers).

ACKNOWLEDGEMENTS

This work has been partially funded by ESA, contract number 16831/03/NL/FF, GNSS Dynamics Simulator and AGGA-4 Test and Simulation Tool (GSTST project).

REFERENCES

1. Silva, J.S., Lopes, H.D., Peres,T.R., Vasconcelos, J.M., Coimbra, M.M., Freire, P., Palomo, P., Pérez, J., Pulido, J.A., García, A., Roselló, J. An Integrated and Cost-Effective Simulation Tool for GNSS Space Receiver Algorithms Development. In proceedings of the ION GNSS 2013 Conference. 16-20 September 2013, Nashville, Tennessee (USA).
2. J. Rosello, P. Silvestrin, J. Heim, "AGGA-4: Core Device for GNSS Space Receivers of This Decade", NAVITEC 2010, December 2010, ESTEC, Noordwijk, The Netherlands.
3. Terma Space, "Terma Emulator Suite", retrieved dec-2013 from http://www.terma.com/media/152081/emulator_suite.pdf
4. Pidgeon, A., Robison, P., McCellan, S., "QERx: A High Performance Emulator for Software Validation and Simulations", DASIA 2009, Istambul, Turkey.
5. Aeroflex Gaisler, "TSIM ERC32/LEON System Simulator" (Product Sheet), retrieved dec-2013 from http://www.gaisler.com/doc/tsim_product_ sheet.pdf.
6. João S. Silva, et al., "The GRANADA Factored Correlator Model Blockset: A Tool for Fast GNSS Receiver Signal Processing Simulations", NAVITEC 2008, ESTEC, Noordwijk, the Netherlands, December, 2008.
7. Global Navigation Satellite System Receiver for Atmospheric Sounding (GRAS), retrieved Jan-2014 from http://www.esa.int/Our_Activities/Observing_the_Earth/The_Living_Planet_Programme/Meteorological_missions/MetOp/GRAS2.