A HW-SW CO-DESIGNED SYSTEM FOR THE LUNAR LANDER HAZARD DETECTION AND AVOIDANCE BREADBOARDING

Pedro Palomo[(1)], Antonio Latorre[(1)], Carlos Valle[(1)], Sergio Gómez de Agüero[(1)], Miguel Hagenfeldt[(1)], Baltazar Parreira[(2)], Almudena Lindoso[(3)], Marta Portela[(3)], Mario García[(3)], Enrique San Millán[(3)], Yuri Zharikov[(3)], Luis Entrena[(3)]

[(1)]Deimos Space, Ronda de Poniente 19, 28760 Tres Cantos, Spain. Phone/Fax (+34) 91 806 34 50/51.
{pedro.palomo, antonio.latorre, carlos.valle, sergio.gomezdeaguero, miguel.hagenfeldt}@deimos-space.com
[(2)]Deimos Engenharia, Av. D. João II, Lote 1.17.01 - 10º, 1998-023 Lisboa, Portugal. Phone (+351) 21893 3010.
baltazar.parreira@deimos.com.pt
[(3)]Universidad Carlos III de Madrid, Conserjería Edificio Betancourt, c/Butarque 1, 28911 Leganés.
{alindoso, mportela, mgvalder, quique, yzhariko, entrena}@ing.uc3m.es

ABSTRACT

This paper presents the HW-SW co-design approach followed to tackle the design of the Hazard Detection and Avoidance (HDA) system breadboarding for the Lunar Lander ESA mission, undertaken given the fact that novel GNC technologies used to promote autonomous systems demand processing capabilities that current (and forthcoming) space processors are not able to satisfy. The paper shows how the current system design has been performed in a process in which the original HDA functionally validated design has been partitioned between SW (deemed for execution in a microprocessor) and HW algorithms (to be executed in an FPGA), considering the performance requirements and resorting to a deep analysis of the algorithms in view of their adequacy to HW or SW implementation.

1. LUNAR LANDER HDA DESIGN

The present and future needs of the Space exploration missions include increasing requirements for system autonomy, in order to cope with the challenging conditions and scenarios that can be presented in these missions (planetary landing, NEO approach and landing, deep space missions, etc.). This has caused advances in the GNC and HDA technologies devoted to handle these autonomy needs, usually involving the use of heavy processing algorithms related to solving complex algorithmic problems in real-time conditions, as e.g. extraction of relevant information from images obtained during the mission. Typical on-board data handling space processors are not well suited to this type of problems as they have performances much lower than the necessary and thus, alternative solutions must be found and deployed. In this scope, the use of Elegant BreadBoarding solutions is necessary to evaluate and mature these technologies before the most critical steps of the mission. In this paper we will present one of such Breadboardings, developed in the scope of the Lunar Lander HDA system and composed by an advanced Power PC 750 processor (PPC, compatible with equivalent space qualified processor models), supported by a co-processing accelerator FPGA. The presented system design has been performed through a HW/SW co-design process, starting from the (base) functionally validated HDA design which had been developed in previous phases of the mission [1] and using it as an executable specification. During the process, the mentioned HDA system has been deeply analysed and partitioned between SW algorithms and HW algorithms, based in the adequacy of each algorithm to be implemented in HW or SW, together with the analysis of the performance improvements to be achieved by each modification, the non-functional requirements of the breadboarding (e.g. the required interfaces) or the maturity of the design of each specific element of the system. The use of the HW/SW co-design approach was necessary as the results of previous prototyping activities [2] had demonstrated that this type of algorithms could not be executed within its required deadline (at present, 10 seconds for the entire execution of an HDA cycle) by using the current space computation technology (this point was further assessed during the testing activities performed in the project). Thus, the need for some type of accelerating co-processor was confirmed, being the FPGA approach one of the main alternatives for the implementation of image co-processing techniques as those used in the project.

The HDA base design was performed in MATLAB/ Simulink, although it includes as well large portions of legacy or hand-written code. This lead to consider that the best approach for the implementation of the breadboard was twofold:

- The implementation of the SW parts of the system are performed mainly through the use of autocoding techniques (which allow a great degree of flexibility but providing a high quality result), or by directly extracting already existing Legacy C-functions;
- The HW design and implementation has been performed through manual VHDL coding, as there were several reasons to avoid the use of automated coding techniques to HDL, including mainly the need to avoid the autocoding problems caused by the legacy code used in the model and the effective use of the limited FPGA resources, which have been largely consumed even by the very optimised design performed. In addition, the used approach

has eased a higher degree of compliance with the applicable ESA standards and guides.

As a side effect to the decisions explained, the HW implementation of the algorithms has implied in many cases to re-engineer the original algorithms in order to allow their effective implementation in HW, as the original algorithms used were not suitable for it. The modified algorithms were validated with respect to the HDA executable baseline, through the execution of specific Monte Carlo tests, when necessary.

Besides, commercial IP cores have been integrated in the FPGA to perform the functions concerning with the communications, such as the communication with the Power PC computer (cPCI) or with the system sensors (Navigation Camera and LIDAR) through Spacewire.

1.1. HDA Functional Architecture

In summary, the main objectives of the Lunar Lander (LL) HDA system functions are to detect several types of hazards from the lunar terrain (by using images and elevation data -DEM- obtained respectively from the Camera and LIDAR sensors) and to select a landing zone that fulfils the mission needs in both safety aspects and mission performance (e.g. the landing site must be reachable within the manoeuvrability constraints of the spacecraft). The new landing site (LS) will be provided to the LL GNC system through a retargeting command, when appropriate.

In accordance to this description, the HDA can be divided in the following main functions, whose functionality is graphically depicted in Fig. 1:

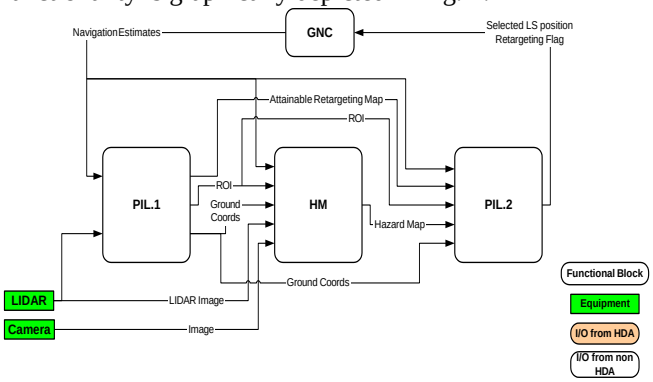

Figure 1: HDA Functional Architecture

- **PIL.1 – Attainable Region**: This function is responsible for determining the area of the terrain that can be reached by the lander (ROI), and associate a reachability score to each LS available inside this area (Attainable Retargeting Map).
- **HM – Hazard Mapping**: Hazard mapping is the process of assigning a hazard score to each LS visible in the sensors' FoV which measures how risky it is for landing. Hazards are computed within the region of interest built by PIL.1 around the candidate landing site.
- **PIL.2 – Decision Making**: This function is responsible for the retargeting decision process. Taking as inputs the reachability and hazard level assessments done respectively by PIL.1 and HM, a decision is made on the best LS, which is provided to the G(N)C system if a retargeting is decided.

1.2. HW-SW Co-design Process

The HW-SW co-design process consists in the allocation to HW or SW of the different functions implemented by the HDA system. As explained in section 1, several factors affect to such decision:

- Functions that directly use data from the sensors are initially allocated to the FPGA, since the sensor data is received there through the system external interfaces. Thus, it has been considered that performing the entire sensor data processing in the FPGA would define a less complex architecture.
- Functions with potentially high processing demand and high parallelization possibilities are allocated to the FPGA. This is typically the case for the IP functions, whose processing can be pipelined by windows, rows, chunks or even pixels.
- Dependencies between functions allocated in different units (i.e. processor and FPGA) should be reduced to avoid communication problems between modules and delays. Two functions strongly coupled are recommended to be implemented together in order to reduce the communication overhead, simplifying also the architecture.
- Complex functions containing decision trees with large number of paths are recommended for their implementation in the processor, as implementing complex decision making into an FPGA can be cumbersome and not efficient (the FPGA implementation of determined algorithms show none or small improvement in comparison with a more traditional software implementation).
- The use of complex mathematical or trigonometric functions is discouraged in the FPGA, even more as the selected FPGA has a limited number of specific resources as those rapidly consumed by the implementation of these functions (e.g. MAC units). Furthermore, the need of programming in a technology-independent VHDL code prevents using some of the most advanced characteristics provided by the selected FPGA and the privative technology-specific implementation of these functions. Also in this category, it is necessary to consider carefully the precision and ranges of the data used and the produced results. The need of implementing Fixed Point solutions into the FPGA can limit the precision of the results obtained, which is particularly important in some type of high-precision algorithms.

The final HW/SW allocation is shown in Fig.2. The functions allocated in FPGA are:

- **Sensor data reception and pre-processing**: Reception and pre-processing of all sensor data (LIDAR and NAVCAM) is performed in the FPGA.
- **Hazard Mapping:** includes Matching, Shadow Mapping, Slope Mapping, Roughness Mapping and Sensor data Fusion functions. These modules are fully embedded into the FPGA, given their intensive use of the images (from camera and LIDAR) and associated high computational cost of the mapping and fusion functions, together with their highly parallelizable characteristics.

The functions selected to be executed on the PPC are:

- **Distance Cost Map (DCM):** this function was initially selected to be implemented in FPGA following the process described above. However, the deep analysis of the function recommended not implementing the algorithm in the FPGA, especially given the type of mathematical operations used. The main reasons to not implement this algorithm in the FPGA are summarized in the following bullets:
  -Some computations do not seem feasible to be implemented in fixed-point format.
  -The module requires a large amount of multiplications to be performed in sequence. For every multiplication there is a loss of accuracy due to truncation. Solutions to reduce the loss of accuracy, such as normalization or using multiple precision arithmetic, are very expensive in terms of resources and cannot be alleviated without strongly penalizing performance, since these operations are in the core of the pipeline processing.
  -Some parts of the module require random addressing, this kind of access to memory typically reduces performance by more than one order of magnitude, destroying all the performance benefits that could be achieved in a FPGA with respect to a microprocessor.
  -The DCM function requires several divisions, square roots and a "tanh" operation, which cannot be efficiently implemented in a FPGA. To avoid performance bottlenecks, they should be implemented in pipeline mode, thus consuming a large amount of resources. Taking into account these considerations, it is quite possible that the FPGA resources would run out.
- **PIL1:** includes Guidance Cost Map and Attainable Retargeting Map computations. PIL1 manages large amount of data, receives several maps and merges these maps in a final map. The analysis of this algorithm shows that its design does not allow a pipelined processing, as it produces the maps above in a fully serial approach, merging the results at the end of the processing and consuming a large amount of resources. In addition, the algorithm implements a series of high precision computations, depending on complex mathematical formulae. In case of need to implement this algorithm in FPGA, it should be analysed and re-engineered in detail in order to adapt it to a HW implementation.

NOTE: It must be remarked that the FPGA cannot store large amounts of data. Therefore, it is assumed that input arrays are uploaded in serial mode and processed in pipeline mode as they arrive. Similarly, output arrays are downloaded in serial mode just as they are produced. Buffering may be used to synchronize pipeline stages and avoid pipeline stalls.

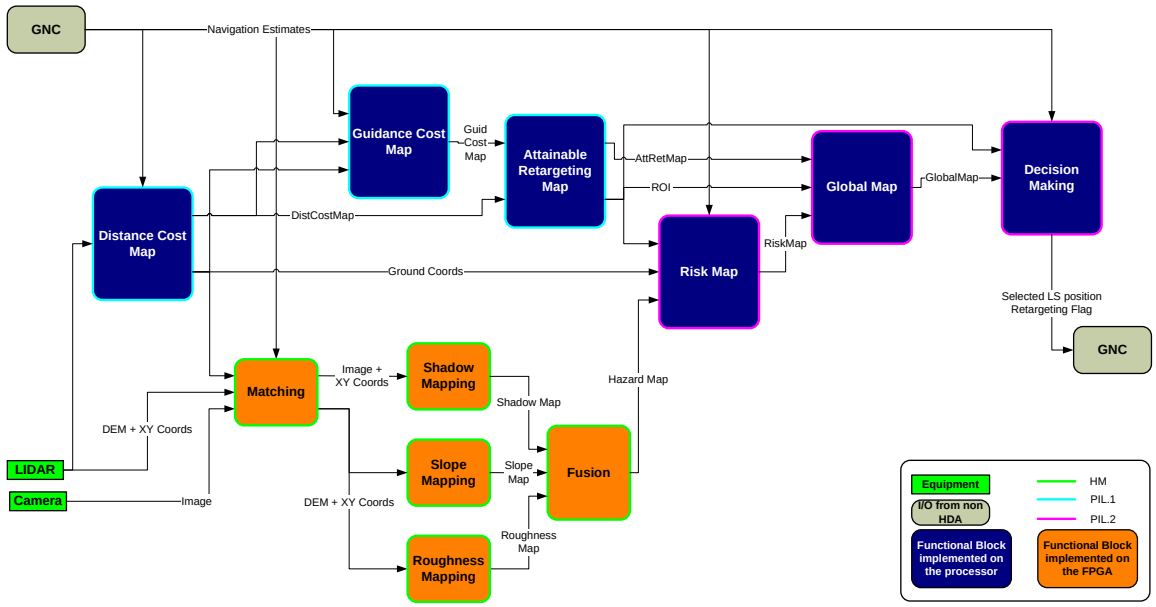

Figure 2: HDA Hardware Allocation

- **PIL2:** includes generation of Risk Map, Global Map and Decision Making. Same justification than PIL1. In addition, the nature of this algorithm as it is implemented at present shows that a pipelined approach is not possible. Further analysis shows also that the PIL2 algorithms will need to be adapted and improved in future phases, which discourages as well the FPGA implementation due to efficiency reasons.

The next step was to define the sequence of activities to perform between the functions allocated in the FPGA and the functions allocated in the PPC in order to obtain the Landing Site and the Retarget Flag, starting since the reception of the Navigation Data and the outputs of the external Guidance and Control element (GC). The sequence of activities, displayed in Fig. 3, shows the nominal execution once the HDA has been initialized and configured. The list of activities listed below are composed mainly by the execution of the algorithms that compose the HDA (see Fig.2) but also activities related to the data handling and interface management:

1. Receive, Store and Forward to GC the Navigation Data: Reception on PPC of the simulated system Navigation Data at 10Hz and forward to G&C.
2. Receive and Store GC Data: GC control data is received and the HDA_Mode is evaluated to decide if the initiation of HDA algorithm has to be performed.
3. RMAP Commands Preparation: The commands to request the acquisition of LIDAR and Camera data are prepared in the PPC and sent to FPGA.
4. Send RMAP Commands: The FPGA sends through dedicated SpaceWire buses the RMAP commands to request the LIDAR and Camera data.
5. Receive and Store Data: The FPGA waits until the end of reception through the SpaceWire buses of the DEM and Image messages (from LIDAR and NAVCAM models respectively) and stores the data in the board's memory, to be used in the PPC side.
6. HDA HW: LIDAR Preprocessing: Convert the LIDAR DEM to Landing Site Frame.
7. HDA SW: Calculate PPC Part. 1: Preprocessing of Navigation data and calculation of the Distance Cost Maps (DCM), which are provided to the FPGA.
8. HDA HW: Calculate Hazard Maps, by using the DCM and the Camera / LIDAR images.
9. HDA SW: Calculate PPC Part. 2: Calculation of the PIL1 (except DCM) in parallel with the calculation on the FPGA (HDA_HW) of the Hazard Maps.
10. HDA SW: Calculate PPC Part. 2: Calculation of PIL2 functions.
11. Send LS: the last step is to send the results of the HDA to the GC, through a dedicated SpaceWire bus.

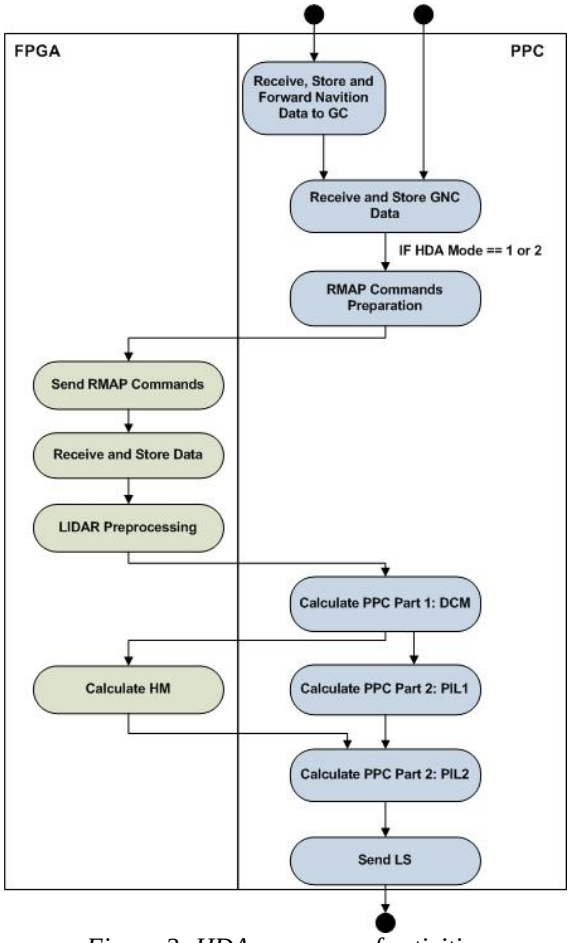

Figure 3: HDA sequence of activities

2. HW IMPLEMENTATION

The HDA Breadboard Hardware architecture and components are presented in Fig.4 including the aforementioned Power PC processor, FPGA and External/Internal interfaces.

The FPGA component is provided through a Gaisler GR development board, equipped with a Xilinx Virtex 4 FPGA and an external SDRAM memory with up to 256 MBytes. This board is connected to the Power PC computer (an ESD development component, representative of the final MAXWELL space qualified processor selected for the final system implementation) through a cPCI interface. SpaceWire physical connectivity is provided directly to the FPGA for reception of the external sensor data (an independent bus for each sensor, which are simulated by dedicated camera and LIDAR models processing image data obtained from PANGU simulator) and to the Power PC system for interfacing to the Lunar Lander GNC.

Fig.4 shows as well the components (both new developments and COTS) included in the FPGA System on Chip (SoC), and communicated through an internal AMBA bus. These components include:

- The SpaceWire Controller module, implementing the logic for proper access to the SpW IP cores.

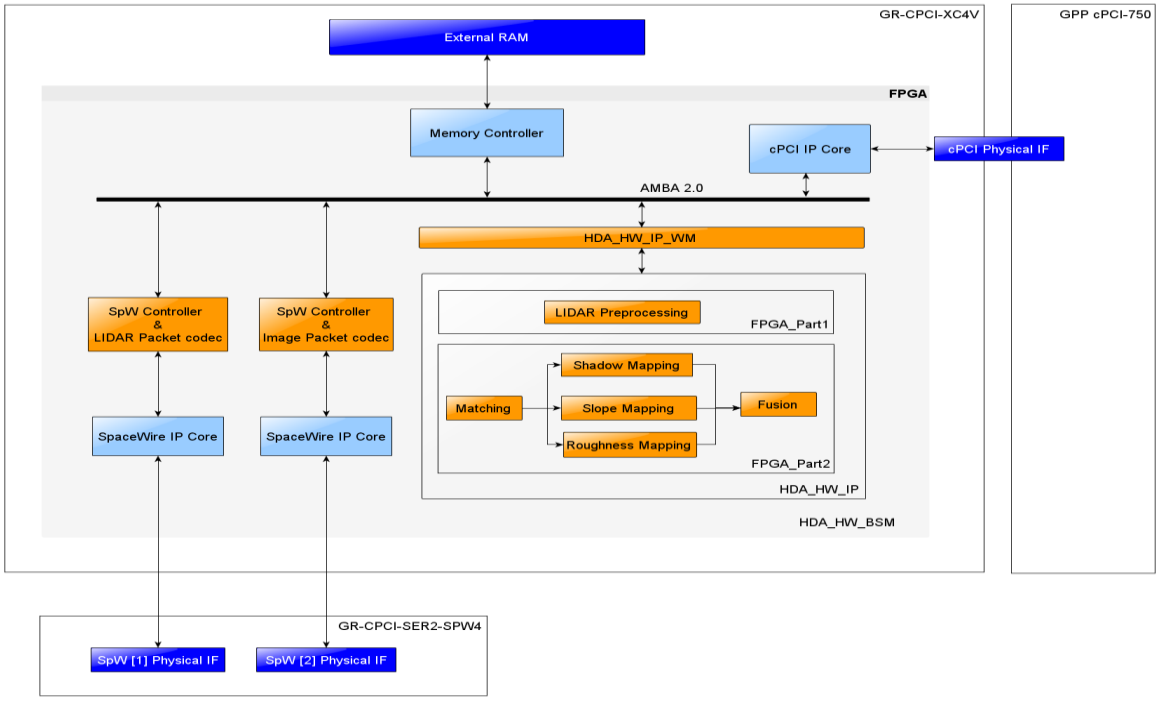

Figure 4: HDA HW System High-Level Architecture

- The IMAGE and LIDAR DMA components in charge of decoding and unpacking the IMAGE/LIDAR DEM, which will be transferred to the Power PC processor through an ad-hoc memory buffer (accessed through cPCI), for PIL processing.
- The Memory Controller and cPCI core manage the interfaces with the RAM memory and cPCI bus.
- The HDA IP core, main development of the project, which implements the HDA functions assigned to Hardware, as described in the previous sections.

The HDA IP Core is connected to the AMBA bus through an HDA IF Adapter module, whose main purpose is to keep the HDA IP Core interface independent of the rest of components and how they are connected (thus, the HDA IP Core is kept portable to other platforms or SoC architectures). The HDA IP core reads the Navigation, IMAGE and LIDAR DEM from the External memory and performs sensor data matching. Afterwards, in parallel, the Hazard Maps are computed. This parallelisation is further increased by the use of processing pipelines in the FPGA, which calculate sections of the maps from the initial images received. The outputs of the different mapping processing are aligned and fed to the Map Fusion component, whose result is finally transferred to the PPC for PIL.2 processing. Intermediate results of the processing chain are stored on memory when needed.
HDA IP Core Control and Status registers are made visible to the PPC, which controls the entire co-processing unit in accordance to the algorithm steps.

Thus, HDA_HW_IP is completely managed by the PPC through the cPCI interface. HDA_HW_IP could also read and write image data to the External RAM.
Communication between the elements inside the FPGA is performed through the use of an AMBA bus compliant with AMBA Bus 2.0 specification, and its AHB and APB buses for data transfer and core control.

3. VALIDATION OF HW IMPLEMENTATION AND RESULTS

The original HDA model has been used as part of the system specification along design phase, together with the HDA closed-loop simulator existing from previous phases. These facilities have been used as well for the validation of the Breadboarding implementation, from unit testing level of the HW and SW modules to the final validation of the system, through the generation of test vectors and golden data results in front of which the system results have been compared. Furthermore, Monte Carlo campaigns of the entire system have been performed in the Simulator in order to validate all specific parts of the code that changed from the algorithmic point of view with respect to the original solution, to accommodate these specific algorithms to alternative numeric algorithms better suited for FPGA implementation. The new alternative algorithms were implemented in MATLAB/Simulink or C code, introduced in the simulator and validated in MC campaigns before go-ahead for VHDL implementation.

Validation of the HW implementation is mainly focused on the HDA_HW_IP Core. The validation phase considers the following main objectives:

- Validate the compliance of the HDA_HW_IP design with the original HDA model implementation.
- Measure the accuracy of the results of the HDA_HW_IP with respect to the original HDA model.
- Measure the performance of the HDA_HW_IP design.

The validation phase included the following tasks, developed in sequence:

- Basic submodule validation: Submodules are tested and debugged before integration in the IP Core.
- HDA_HW_IP Unitary Function Tests.
- HDA_HW_IP Integration Tests.
- FPGA validation: HDA_HW_IP Design validation in the real hardware.

Following a top-down hardware design methodology, the design is thoroughly validated by functional simulation before going into the FPGA implementation. Simulation results are compared to the original HDA MATLAB/Simulink/C implementation.

Simulation Tests

Simulation is the basic mechanism for unitary tests. It is also used for HDA Integration tests when required.

Simulation tests use input data from the HDA baseline implementation and automatically compare results with the original HDA model implementation.

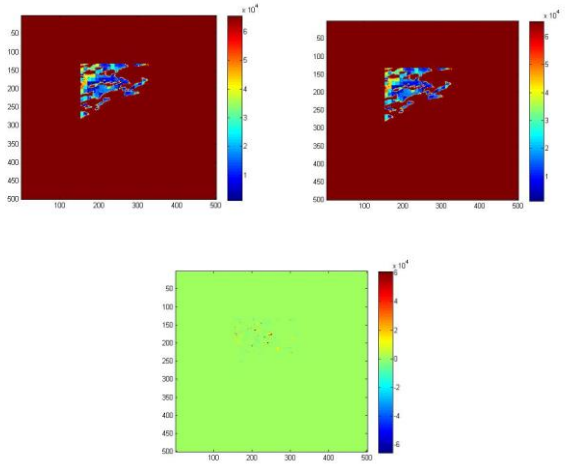

Figure 5: Example of HDA HW output: Variable HAZARD_MAP (far range)

Hardware Tests

Hardware tests are performed upon integration of the HDA IP Core in the FPGA. Hardware tests generally consist on repeating the simulation tests on board. Specifically, hardware tests are required to validate the external FPGA interfaces and to accelerate the execution of large sets of test cases. Fig.5 and Fig.6 show an example of the outputs that HDA HW sends to HDA SW: the variable HAZARD MAP in close range, represented as a map that reflects the overall hazard level of the candidate landing sites. The image in top-left shows the expected results, the image in the top-right shows the results obtained in HW, and the image on the bottom shows the difference between them. The analysis of the results showed that the errors were negligible or located near areas of already identified Hazard (thus, far from the safe landing sites).

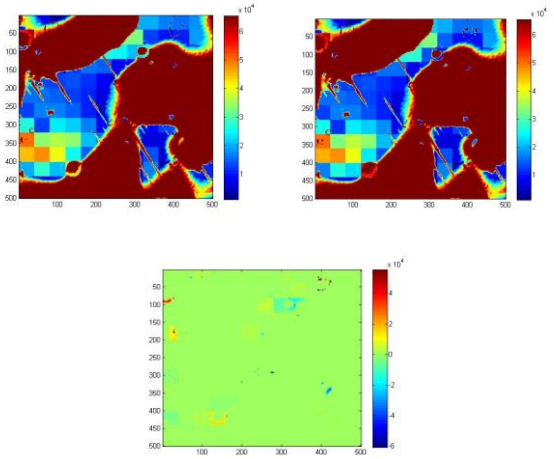

Figure 6: Example of HDA HW output: Variable HAZARD_MAP (close range)

Worst-case error statistics are extracted from the analysis performed, to quantify the effect of the FPGA fixed-point implementation (and other errors) in the overall HDA implementation (i.e. with conversions from doubles to fixed point in the inputs/outputs). These error statistics are then used in the general HDA testing campaigns, performed through Monte Carlo simulations, ensuring that 99% LS decisions are correct.

4. SW IMPLEMENTATION

The class diagram shown in Fig.7 *r*epresents the High Level Architectural Design for the HDA SW component, allocated to the system Power PC processor and run on top of the VxWorks operating system. This component is in charge of performing the HDA SW activities, both related to performing and controlling the HDA algorithms and to the additional functions necessary to communicate with the Lunar Lander GNC computer or the Navigation data provider, and providing debug or control data. Three main groups of modules or layers are shown:

- **HDA Layer Service:** The module is in charge of managing the external communications of the system, managing the hardware drivers and independent to the rest of the Software functions from the underlying interface hardware. This layer is composed by sporadic tasks that waits the

reception of the inputs data from the Navigation simulator and from the Guidance and Control

- **HDA Application:** The module is in charge of controlling the HDA execution and managing the application data necessary to produce a landing site retargeting, estimated navigation data and the data sets exchanged between the HDA SW and HW. This layer is in charge of managing the system data flow and controlling the execution of the HDA algorithms. It waits the reception of the simulated Navigation data and G(N)C data and analyses them in order to command the HDA process execution; First, it obtains feedback information from the G(N)C and Navigation, then it checks if the data indicates that shall be calculated the Landing Site. Once the HDA has been started, this layer is in charge of commanding the execution of the HDA SW, sending the results of the HDA SW to the FPGA through the PCI and commanding the HW to start the processing. When the HDA HW finalizes the processing of the data, the HDA Application will command the retrieval of the data results, providing them to the HDA SW to continue its activities and confirm the landing site or calculate a new possible, which will be sent to the G(N)C through the SpaceWire interface.
- **HDA SW:** The module that performs the Hazard Detection and Avoidance algorithms and drives the execution of the component, or application Software Component (SWC). These functions derive from the HW/SW co-design, where the HDA functions were partitioned into HW and SW.

The HDA SW activities are assigned to independent tasks depending on the Software design, frequency requirements and concurrency needs.

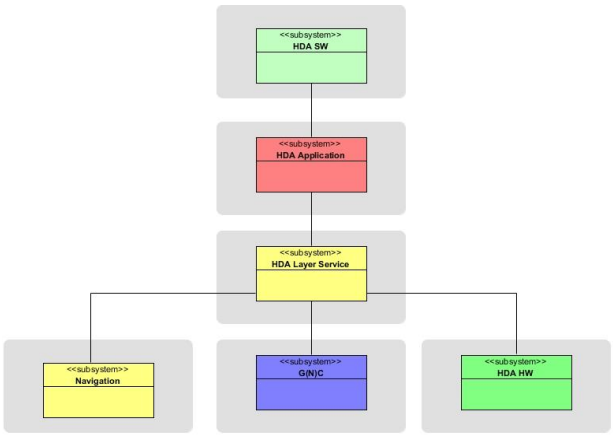

Figure 7: HDA HW System High-Level Architecture

The scheduling analysis of the system has provided the inputs necessary to perform this assignment, with the main objective of meeting the real time requirements of the system such as the HDA execution time deadlines. Synchronization mechanisms are used as well when necessary to assure the integrity of the data sets, which can be compromised by the simultaneous access of more than one task.
The synchronization between the FPGA and the PPC has been kept as simple as possible, prioritising asynchronous signalling over clock-based synchronization.

5. SYSTEM VALIDATION AND RESULTS

The HDA validation approach is based in the injection in open loop of pre-recorded inputs (previously generated in the HDA functional simulator or FES) for different test cases / reference trajectories, studied to cover the different situations that can be faced by the system: since the simple trajectory with no retargeting needs, to more complex test cases where several new LS are selected. For the final validation steps, three reference scenarios were selected (Nominal spacecraft mass, Maximum mass and Minimum mass). In these scenarios the HDA is executed in two different phases:

- **Far range phase:** Far range zone with ranges between 2500m and 1500m. The LIDAR model is parameterized for far range conditions, i.e. ground resolution of 2.5m.
- **Close range phase:** Close range zone with ranges between 300m and 150m. The LIDAR model is parameterized for close range conditions, i.e. ground resolution of 0.25m.

The process of HDA validation is split for three sets of test campaigns:

- Unitary tests of the HDA: Performed to ensure that BB implementation of the HDA units (HW and SW) is correct, and later to check if integration into complete BB environment is correct. Reference data generated for unitary and integration tests are the same, with the difference that during (step-wise) integration tests the inputs are generated in BB environment. Although HDA is previously validated in functional engineering simulator using NAV performance model (Astrium), the approach for unitary/integration tests is done using ideal navigation inputs (as it simplifies the analysis process of internal HDA algorithms).
- Integration test for the HDA-HW and HDA-SW: Performed to ensure that HDA BB implementation of the HDA HW and HDA SW is correct.
- Integration tests of the HDA/G(N)C: Performed to ensure that the whole BB HDA implementation is correctly integrated in BB environment. Reference data is generated on functional engineering simulator using NAV performance model (Astrium) and implementing sensor delays

The outputs produced by the entire HDA system are compared with the outputs produced by the HDA functional / reference model in order to perform the following validations:

- Real time functional performance validation: This validation is based on comparison of the results generated by the HW/SW functional design model respect the Hardware/Software implementation results, in an open loop testing approach. The "functional performance" is measured by comparing the results of the HDA (mainly the newly selected landing sites) and evaluating the differences obtained. Fig.8 shows an example of correct HDA decisions (safe and with 2.7 footprint margin) over a map with safe and unsafe zones.
- Resource performance validation: The performance of the HDA system in real time and resources aspects is measured by the computation of several budget figures (different in Hardware and Software), which include the Worst case execution time (WCET) of the composed HW/SW system, for the Nominal and Worst-case reference test cases defined. This WCET includes the complete processing since the start of the HDA cycle until the production of a final result (and passing through the sensor commanding and sensor data acquisition and processing, the computation of the hazard maps, the piloting functions, etc.). The total HDA execution times including all the activities defined in Fig.3 are shown in Tab. 1 below, in seconds.

Table 1: HDA Timing

Test Case	Far (sec)	Close (sec)
Nominal spacecraft mass	7,4828	9,6368
Maximum spacecraft mass	7,481	8,973
Minimum spacecraft mass	7,4826	9,4816

The same approach has been followed to validate the unitary implementation of the HW and SW modules, as well as the integration of the independent parts.

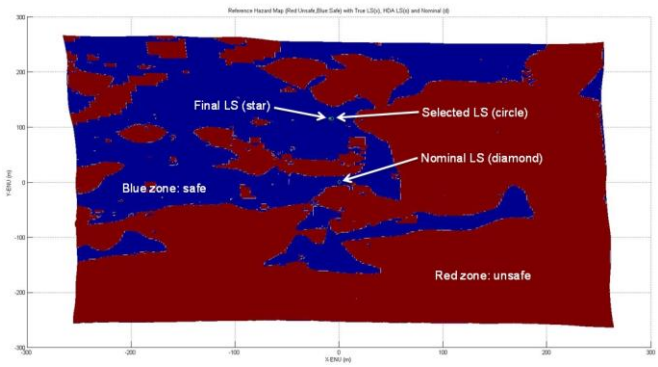

Figure 8: Example of HDA final decision

6. CONCLUSIONS

The precise analysis and later implementation in hardware of specific parts of the functional Lunar lander HDA algorithms has allowed a significant reduction of the system execution time, while keeping results accuracy. The total WCET of the HDA is around two seconds in the entire execution of the HDA HW which is a dramatic improvement from the results of the execution of the same algorithms in SW. However, along the development of the project it was also demonstrated that not all the SW is adequate for HW implementation, due to feasibility, efficiency or complexity reasons which had to be analyzed with care.
The implementation of the HDA algorithms in a HW/SW co-designed approach, using the system model as executable specification and validation platform, has proved as a successful solution from the point of view of both, performance and functional results.
Finally the BB activity has increased the TRL level of the HDA subsystem to TRL 4, or TRL 4/5 for the integrated G(N)C+HDA subsystems correctly operating together in a HW and navigation simulated environment (i.e.: simulated image / DEM generation, obtained through PANGU tool and camera/LIDAR models).

As final conclusion HW/SW co-design and partitioned implementation has been demonstrated to be a good solution for certain systems which involves the use of heavy processing algorithms related to solving complex algorithmic problems in real-time conditions, but it is important remark that this process demands a complex and costly effort, which should not be underestimated.

ACKNOWLEDGEMENTS

This work has been partially funded by ESA, contract number 4000101533 LUNAR LANDER - PHASE B1**,** and by the Spanish CDTI as part of the PERIGEO project, file number IPT-20111022**.**

REFERENCES

1. Parreira, B., Vasconcelos, J.M., Montaño, J., Peñin, L.F., Hazard Detection and Avoidance in ESA Lunar Lander: Concept and Performance. In proceedings of the AIAA GNC 2013 Conference.
2. E. Kervendal, G. Flandin, K. Kanani, J. Morand, B. Parreira, A. Caramagno, J.C. Bastante, J. Quirce, J. Dinis, P.Motrena, J. Rebordao, C. Philippe, Vision-Based Hazard Detection and Avoidance for Martian and Lunar Landing: the HASE study for ESA. In proceedings of the GNC 2011 Conference.