HARDWARE IMPLEMENTATION OF THE SPOT PAYLOAD FOR ORBITING OBJECTS DETECTION USING STAR SENSORS

Mohamed Salim Farissi[*], **Ivan Agostinelli**[*], **Marco Mastrofini**[*], **Fabio Curti**[*], **Cosimo Marzo**[†], **Claudia Facchinetti**[‡], **Luigi Ansalone**[‡]

[*] *School of Aerospace Engineering Sapienza University of Rome, Rome 00138, Italy.*
Email: mohamedsalim.farissi@uniroma1.it
[*] *School of Aerospace Engineering Sapienza University of Rome, Rome 00138, Italy.*
Email: ivan.agostinelli@uniroma1.it
[*] *School of Aerospace Engineering Sapienza University of Rome, Rome 00138, Italy.*
Email: marco.mastrofini@uniroma1.it
[*] *School of Aerospace Engineering Sapienza University of Rome, Rome 00138, Italy.*
Email: fabio.curti@uniroma1.it
[†] *Centro di Geodesia Spaziale "Giuseppe Colombo", Italian Space Agency, Matera, 75100, Italy.*
Email: cosimo.marzo@asi.it
[‡] *Italian Space Agency, Via del Politecnico,00133, Rome, Italy.*
Email: claudia.facchinetti@asi.it
[‡] *Italian Space Agency, Via del Politecnico, 00133, Rome, Italy.*
Email: luigi.ansalone@asi.it

Abstract

Space debris issue has become an attractive challenge for many applications in the framework of Space Situational Awareness (SSA) and Space Surveillance and Tracking (SST). The Star sensor image on-board Processing for orbiting Objects deTection (SPOT) fits in this field as an innovative space based autonomous and versatile system for Resident Space Objects' optical detection via star sensors and for different Earth orbits scenarios. This system is planned to be a payload for an In-Orbit Validation (IOV) activity in the next future. The purpose of this paper is to show the architecture of the SPOT system together with its implementation on a System on Chip (SoC)/Field Programmable Gate Array (FPGA) space representative board. The SPOT algorithms involve several layers of filters which are relatively expensive in terms of computational latency, limiting their applicability to real-time image processing applications. This work presents the design and implementation of SPOT algorithm on the Zynq-7000 SoC using Xilinx FPGA and ARM CPU. Algorithms have been modelled with Simulink and implemented on FPGA using Xilinx system generator with aiming to optimize both processing time and area usage. A Hardware-In-the-Loop (HIL) setup was developed as well, to verify the performances and robustness of the SPOT algorithms and simulating critical scenario by using real night sky images from acquisition campaign.

Keywords: Space Debris, Star Sensors, FPGA, SSA, SST, HIL

1. Introduction

Space debris has been a threat since the late 1990s, when the Space Shuttle mission was a reality and the first module of the International Space Station was put into orbit. Since then, numerous missions for scientific, commercial, environmental and national security purposes have been launched. This has led to a huge amount of resident space objects (RSOs) orbiting the Earth.

They are made up of space debris and active / inactive platforms. Due to the need to monitor this phenomenon, the creation, maintenance and updating of the RSO catalog is mandatory. Nowadays, only the United States and Russia have operational space surveillance systems with a regularly updated catalog of space objects, while Europe is developing its own space surveillance and tracking systems (SST) to achieve the same capacity.

Radars are typically used for tracking objects in low-Earth orbit (LEO), and optical systems are mainly used for GEO surveillance [1]. These ground-based SST sensors are mainly used nowadays but they have limitations:

- Radars are limited by power consumption constraints but can guarantee a 24-h coverage for RSOs detec-

tion and monitoring.
- Optical Telescopes are limited by weather conditions and can work just during the night.

Space missions for SST purposes are currently being studied and developed [2], In the period 2017-2020, one of the main goals of the ESA's SSA program was the development of space sensors. The reason for this choice is to go beyond the limits of sensors on ground and increase coverage to detect SARs. In particular the ability to detect small objects due to the narrow distance between the target in orbit and the observer in orbit.

This topic is one of the most recent challenges in the field of SSA and a lot of research is being carried out. Indeed, Airbus was in charge of the feasibility study, design and development of a space-based space surveillance mission (SBSS)[3]. Here, the platform is able to periodically scan the entire GEO belt with a low field of view (FOV) and with an appropriate low phase angle (achieved through an anti-sun pointing strategy). This will be achieved with an on-board telescope with an aperture of about 20 cm which is sufficient for SST tasks. Other SBSS missions have been launched in the past such as the Mid-Term Space Experiment (MSX) and the Canadian NEOSSat (Near-Earth Object Surveillance Satellite) [4].

The main innovative idea of the SPOT project is the use of on-board star sensors, optical devices whose main objective is attitude determination. Their image data can also be used for object detection by developing an appropriate image processing and RSOs information extraction payload. In this way, creating a constellation of space-based observers for SST purposes will be quick and cost effective, with all the benefits for the SSA community and future space missions.

The objective of this article is the description of the SPOT architecture and its implementation on representative target hardware. The article is organized as follows: Section 2 describes the architecture of the algorithm and the logic of each module. Section 3 is the implementation of the algorithm on the SoC/FPGA board with all the techniques and methods applied. Section 4 focuses on the night sky image acquisition campaign for the collection of test input images. Section 5 presents the SPOT unit HIL tests and the results with a validation criterion to assess its correct functioning. In section 6, conclusions and future goals are provided.

2. Methodology

2.1. Architecture

The on-board SPOT unit has to detect orbiting objects starting from optical measurements from star sensor. An object appears as groups of pixels spreaded over the detector, according to its size and velocity with respect to the observer. In normal mode, during the image analysis process, SPOT can recognize objects (stars or RSOs) in the Field Of View (FOV) of the sensor and if in Active Mode (AM), it can exclude a large part of the stars filtering them out with a technique very similar to those used during usual star tracking operation. The AM depends on the availability of the attitude information from the hosting platform.

This section summarizes all the pre and post processing operations to obtain the unit vectors of the detected objects in the hosting platform's reference frame, starting from the raw images until the centroids' coordinates computation of the desired objects. Figure 1 shows the overall ON-BOARD SPOT Architecture. In particular, SPOT can work with a single image or with a continuous flux of images from the star tracker. As can be seen in the figure, with a single image the Cluster Fusion module is skipped and, as will be explained in the following, the "tracking operation" of the Cluster Growth module is not considered.

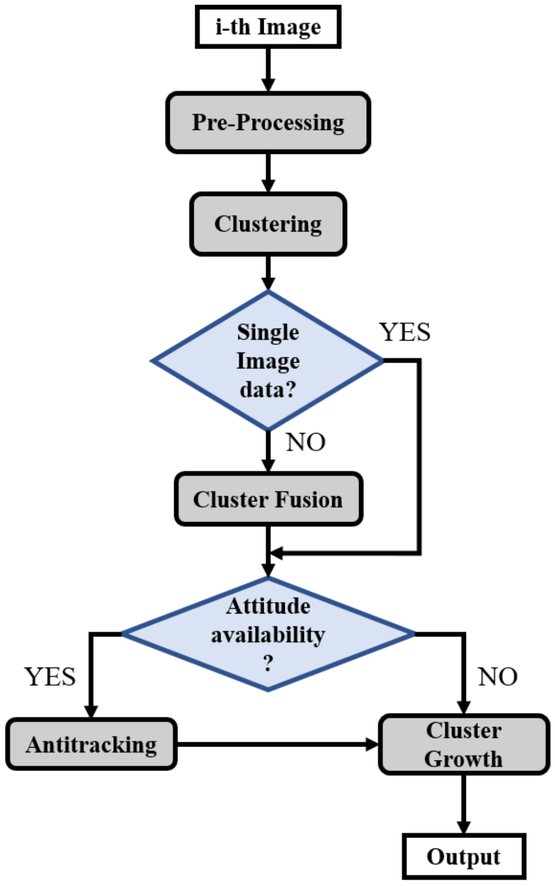
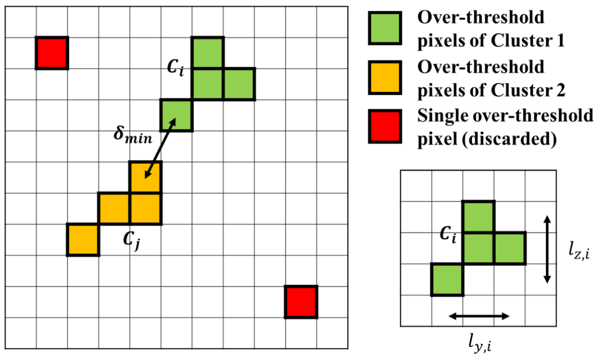


Figure 1: ON-BOARD SPOT Architecture.

2.1.1. Pre-processing. The raw image from the star tracker is processed by the "Pre-processing" module. The high-energy pixels belonging to stars' or RSOs' signals must be distinguished from the low-energy pixels relative to noise. Hence, the Pre-processing module selects the pixels whose signal-to-noise ratio is greater than a user-defined threshold and discards the useless information. In particular, this pre-processing is called *Segmentation* and it is based on a dynamic approach for the background noise estimation consisting in the comparison of the signal of each pixel with the average signal of its local neighborhood. Pixels are considered for further processing when their values are greater than the background noise (BKG_0) plus a threshold (τ_{pre}). Therefore, only segment information is considered for further analysis relative to objects' detection. In particular, the Pre-processing outputs are the pixels' coordinates of each detected segment $\mathbf{p} = [p_y, p_z]^T$ and their energy $E(\mathbf{p})$, obtained subtracting the background noise to the signal intensity of the pixel.

2.1.2. Clustering. The Clustering module has to recognize all the neighboring pixels belonging to individual objects. Usually, star sensors carry out a simple clustering algorithm (called here as *Primitive Clustering*) but it is not enough when dealing with fast objects. A "primitive cluster" is defined as groups of pixels which share at least one corner, as shown in Fig. 2, while single pixels are mostly related to noise or too faint objects and they are automatically discarded and not considered for the following analyses.



Over-threshold pixels of Cluster 1
Over-threshold pixels of Cluster 2
Single over-threshold pixel (discarded)
C_i
δ_{min}
C_j
C_i
$l_{z,i}$
$l_{y,i}$

Figure 2: Example of two primitive clusters and discarded single pixels.

The green pixels in the figure share an edge or a corner and are associated to the same cluster (named "Cluster1"). The same can be said for the orange pixels ("Cluster2"), while the red ones are discarded representing noise or too faint objects.

Under dynamic conditions, objects in the FOV can generate broken streaks. It means that pixels associated to the same object can be spreaded into several small and distinct clusters. For this reason, a suitable technique has been developed in order to relate to the same object (star or RSO) the different primitive clusters of the broken streak. It is important during this operation to avoid wrong primitive clusters agglomeration coming from noise or wrong matching between different objects. This technique is called *Improved Clustering* and it is based on three filters ([5]):

- *Minimum distance filter*: The distance between clusters δ_{min} is evaluated as minimum pixel-by-pixel distance using the uniform norm and must be lower than a user-defined threshold ε_{dist} (Figure 2). The minimum distance condition is defined as:

$$\delta_{min} \leq \varepsilon_{dist} \tag{1}$$

- *Increasing length filter*: Let $\mathbf{l}_i = [l_{y,i}, l_{z,i}]$ be the length vector of the i^{th} Cluster (C_i), collecting the horizontal and vertical projections of the cluster. To merge the clusters C_i and C_j, the merged cluster C_ij must satisfy precise conditions related to $\mathbf{l}_i$, $\mathbf{l}_j$ and $\mathbf{l}_{ij}$.

- *Density filter*: Clusters C_i and C_j can be merged only if some necessary conditions related to their densities are satisfied. In particular, the density of the i^{th} cluster is defined as:

$$d_i = \frac{N_i}{\lambda_i} \tag{2}$$

where, N_i is the number of pixels of C_i and λ_i represents a virtual length of the cluster. Clusters with low density values are likely to be broken into different pieces.

At this point, some primitive clusters have been merged and the cluster centroids are computed using the pixel coordinates $\mathbf{p} = [p_y, p_z]^T \in C_i$ throughout an energy-weighted average:

$$\mathbf{c}_i = \frac{\sum_{j|\mathbf{p}_j \in C_i} \mathbf{E}(\mathbf{p}_j) \cdot \mathbf{p}_j}{\sum_{j|\mathbf{p}_j \in C_i} \mathbf{E}(\mathbf{p}_j)} \tag{3}$$

where $\mathbf{E}(\mathbf{p}_j)$ is the signal intensity of the pixel $\mathbf{p}_j$.

2.1.3. Cluster Fusion. To compare two successive images and merge the objects which are supposed to be the same in the two frames, the Cluster Fusion module was developed. This module is designed to work only if two consecutive images are available, otherwise it is skipped (Figure 1). It cannot be applied if a non-negligible time interval separates the first and second image. The required operations are very similar to the ones described for the Improved Clustering. The main difference is that the clusters C_i and C_j which are compared belong to two different images. The minimum distance filter and the density filter are applied. The output of the Cluster Fusion does not contain not-merged clusters, i.e. clusters that do not appear in both the successive images. In this way, it is possible to recognize the persistence of an object in the frames and allows the software to understand the direction of the moving object in the FOV.

2.1.4. Antitracking. The Antitracking module is required in order to remove stars from the set of detected objects. This module requires as input the current attitude quaternion, a star catalogue (e.g., Hipparcos Catalogue) and the sensor's characteristics. It is an operation very similar to the technique used during usual star tracking operation for star tracking with the difference that all the objects recognized as stars are filtered out. Antitracking removes stars according to the following steps:

1) Evaluate 2D centroids in 3D camera frame. This operation requires sensors' characteristics.
2) Evaluate the 3D centroids in the inertial frame using the current quaternion.
3) Compare the centroids in the inertial frame with catalogued stars directions.
4) Remove centroids which report angular distances from catalogued stars greater than an user defined threshold. The adopted threshold depends on quaternion accuracy.

2.1.5. Cluster Growth. The Cluster Growth algorithm is used to identify and track objects in the star sensor's FOV. This module is applied to build an on-board database of tracked objects (stars or RSOs). It has to compare and analyse successive couples of images. The goal is to track RSOs from their appearance in the FOV until they leave it and save their positions and time instants of each couple of images. The output of this module represents the final output of on-board SPOT unit. The on-board data structure is transmitted on ground for post-processing operations and orbit determination from too short arcs. At the initialization of the Cluster Growth, the merged clusters returned by the Cluster Fusion are saved as independent objects. When a new couple of images is analysed and the fusion operation is performed, the new merged clusters are compared with the previous objects and if a cluster is recognized as belonging to an already detected object, it is updated with the upcoming couples. When there is no correspondence, after a maximum number of couples defined by the user-defined parameter N_{jump}, no update occurs. Contrary, merged clusters not recognized as belonging to previous objects lead to the creation of a new object in the on-board database.

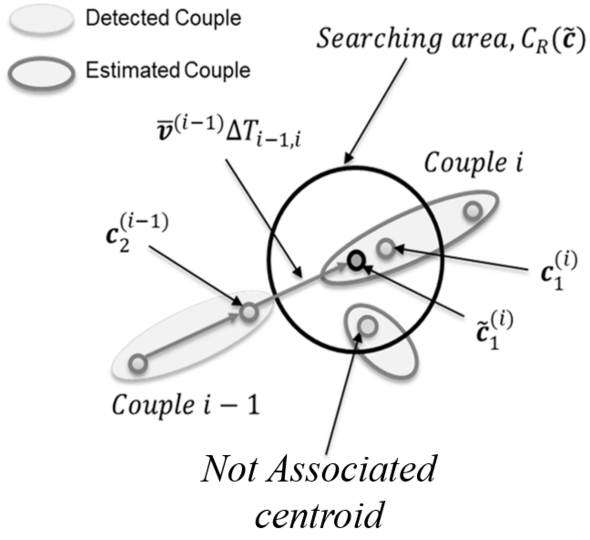

Detected Couple
Estimated Couple
Searching area, $C_R(\tilde{\boldsymbol{c}})$
$\bar{\boldsymbol{v}}^{(i-1)}\Delta T_{i-1,i}$
Couple i
$\boldsymbol{c}_2^{(i-1)}$
$\boldsymbol{c}_1^{(i)}$
$\tilde{\boldsymbol{c}}_1^{(i)}$
Couple $i-1$
Not Associated centroid

Figure 3: Cluster Growth algorithm.

The filtering operation is based on three separate steps:

- *Position filter*: The core of the algorithm is based on the estimation of the centroids' positions in the new couple of images for each tracked objects. The distance-based criterion of the Cluster Fusion must be extended to associate the merged clusters of the first and second couples. Indeed, the merged clusters are not as close as in the case of the Cluster Fusion and an estimate of the cluster position in the successive couple of images is required. Using velocity estimations computed as a first order finite difference during the fusion operation up to the $(i-1)^{th}$ couple of images, the average velocity $\bar{\mathbf{v}}^{(i-1)}$ is computed considering the number of available images N_a. With this information and the last saved position of the centroid, the estimation of the centroid position $\tilde{c}_1^{(i)}$ in the first image of the i^{th} couple is evaluated considering the centroid position of the second image in the $(i-1)^{th}$ couple of images and the estimated displacement of the centroid. The centroid search is performed if and only if $\tilde{c} \in D$, where D is a circular neighbourhood of the centroid estimation, called *Searching Area* (Figure 3). If one or more detected centroids fall inside D, they are selected as candidates for the tracked object.

- *Velocity filter*: The position filter represents a first preliminary selection of feasible candidate centroids, but more than one centroid is likely to fall inside the searching area. The velocity filter refines the previous results by comparing the velocity of candidate centroids with the velocity of the tracked object. The filter considers both the direction and the magnitude of the velocities. Defining ϕ as the angle between the two directions of motion, the objects characterized by angles greater than a user-defined threshold ϕ^* are discarded.

- *Confirmation phase*: If more than one centroid has been detected by the previous filters, a criterion to choose the best candidate must be accounted for. To assess which is the most reliable centroid, the estimates of the cluster density and centroid displacements are included in a last filter based on a minimization criterion. Indeed, a function F is introduced:
$$F = F_{dist} + F_{dens} \tag{4}$$
where, F_{dist} is related to the difference between the estimated displacement of the centroid and the calculated one and F_{dens} is based on the object density and it is evaluated as a normalized difference. The candidate with the lowest value of F is chosen.

At this point, the centroids are associated to the considered object and the on-board database is updated [6].

2.2. Hardware description

This work presents the design and implementation of the SPOT algorithm on the Zynq-7000 SoC. This circuit is based on an ”AP SoC” (All Programmable System-on-Chip) architecture, which integrates a dual-core ARM Cortex-A9 processor associated with an FPGA from Xilinx. Hence, it allows software and hardware development of applications. The algorithms are developed and implemented following the Co-Design methodology where the design flow should be divided across the two implementation platforms, FPGA and Microprocessor with the intention of benefiting from each of their strengths to share the processing load. The parallelism nature of the FPGAs enables the capability to run several processing elements in parallel, that it would be possible to process the data during few clock periods. This feature makes FPGAs suitable for numerous high-performance applications that require intensive and high-speed computations like image processing. While Microprocessors are performing well with managing and controlling data as well as decision making. In this case, the Microprocessor will be used as a ”master” configuration unit to direct the flow of data to the ”slave” FPGA device.

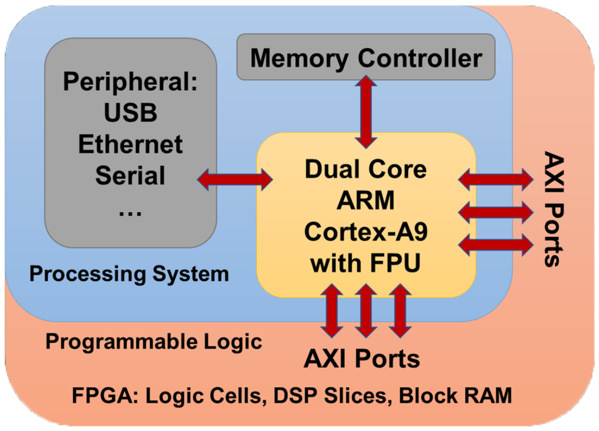

Memory Controller
Peripheral: USB Ethernet Serial ...
Dual Core ARM Cortex-A9 with FPU
AXI Ports
Processing System
Programmable Logic
AXI Ports
FPGA: Logic Cells, DSP Slices, Block RAM

Figure 4: Zync Hardware Architecture.

It must be mentioned first, before proceeding, the Central Processing Unit (CPU) implementation will not be covered here as the purpose of this paper is to only show the implementation of SPOT units that are completely implemented on FPGA. The FPGA must contain enough programmable logic block for processing and enough memory for data holding to be able to support the processing of all SPOT units.

2.2.1. FPGA. This paper describes an efficient FPGA-based hardware design for SPOT algorithms. These algorithms involve multiple layers of filters that are relatively expensive in terms of computing latency, limiting their applicability to real-time processing on serial processors such as CPUs. In order to meet the real-time requirements needed where high-speed parallel data processing is requested, the SPOT filters are well suited for a hardware implementation on FPGAs which can dramatically increase performance per watt in comparison to the equivalent software implementation taking the advantage of their parallelism in application execution. With parallel computing multiple processing elements executing a sequence of instructions at the same time allowing to run many functions at once, and therefore get results more quickly. FPGAs contain an array of programmable logic blocks, and reconfigurable interconnects to link the logic blocks to each other while retaining the flexibility to be reprogrammed to implement different logic functions at any stage during and after the design process. Programming an FPGA is a process of customizing its resources to serve different purposes. This involves modeling the program instructions using configurable logic blocks and others to perform complex functions, or basic logics like ”OR”, ”AND” and dedicated multiplexers.

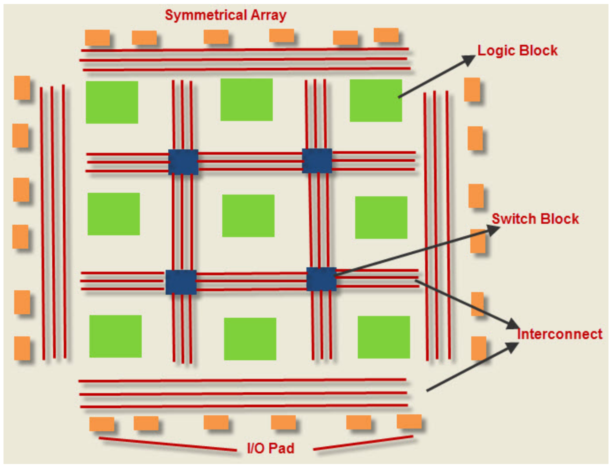


Figure 5: FPGA architecture description.

3. Implementation approach

3.1. System Generator (software)

Generally, FPGAs are programmed with a Hardware Description Language (HDL) like Verilog, SystemVerilog, or VHD. With HDLs, the designer describes through logic and instantiating modules to use to implement the algorithm. To implement SPOT algorithms using HDLs requires thousands of coding lines, which is impractical and time-consuming. An alternate solution is using Xilinx System Generator, coupled with a graphical interface under the MATLAB-Simulink that enables the use of the MathWorks model-based Simulink design environment for FPGA design makes it very easy to work with in comparison to the other software for hardware description. The library of Xilinx System Generator includes many building blocks, allowing faster prototyping and design from a high-level programming standpoint. As a result, designers can define an abstract representation of a system-level design and easily transform the algorithms into a gate-level representation. Another benefit of using the Xilinx System Generator for the hardware implementation is that it allows the FPGA module to be co-simulated with the test vectors provided by MATLAB Simulink Blocks. In Software co-simulation, all Xilinx blocks are connected between ”Gateway In” and ”Gateway Out” blocks, which respectively behave as input and output for the hardware design.

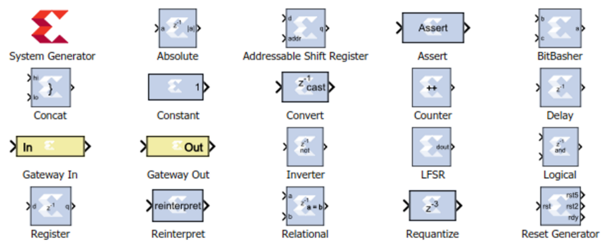


Figure 6: Xilinx blockset in simulink

3.2. Fixed point arithmetic

Normally, floating-point implementations require larger amounts of FPGA resources. This increased use of resources results in higher energy consumption and, ultimately, an increase in the overall cost of implementing a design. Therefore, reducing the use of FPGA resources inherently leads to lower power consumption and enables massively increased computing capabilities within the FPGA. Again, converting from a floating-point design to a fixed-point design can significantly save power and area efficiency while maintaining the same level of precision and comparable performance. In some cases, the results can even be improved. To meet these challenges, it is necessary to thoroughly evaluate the lower precision (fixed point) implementations of MATLAB codes before targeting the FPGA implementation.

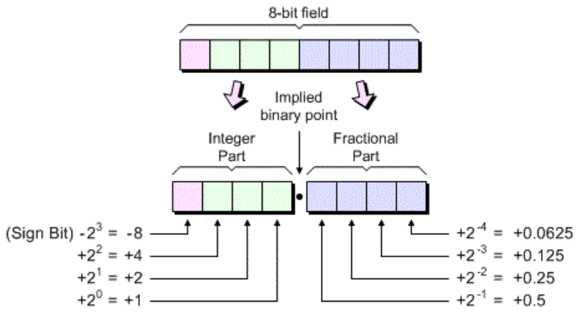


Figure 7: Fixed point arithmetic description.

3.3. Pipelining

One important objective to be taken into account during the design is to increase the clock rate and throughput of an FPGA. This is achieved by pipelining design mechanism. Pipelined designs take advantage of the parallel processing capabilities of the FPGA to increase the efficiency of sequential code. For this, the code is divided into several small parts and then separate them using registers. With the pipelining, the processor can start executing a new input without waiting for the previous one to complete.

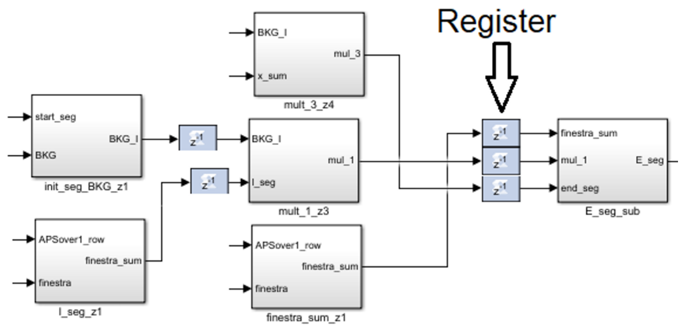


Figure 8: Design with Pipelining approach.

The different processors are separated by buffer registers, which are all linked to the clock signal. At each clock cycle, the registers become accessible for writing, which causes the data to pass to the next step.

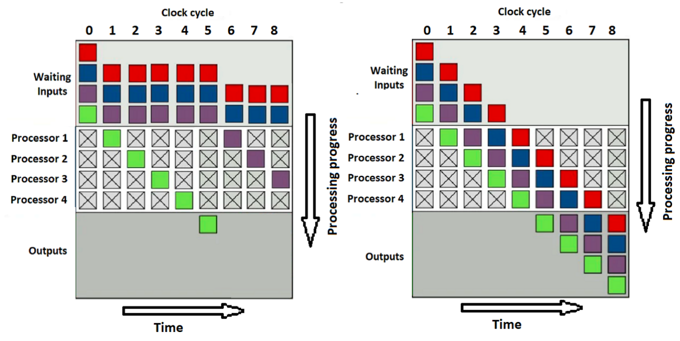


Figure 9: Non-Pipelining process (left) and Pipelining process (right)

In the non-pipelined design (figure on the left), as each input occupies all four processors till the output is produced, since each processor takes one clock cycle, all four inputs will take twenty clock cycles to be fully processed. In the pipeline design (figure on the right), the input accesses the processor row as soon as it is free. Therefore, the output is produced for each clock pulse starting from the fourth clock cycle. Indeed, each input must pass through four registers during its processing before reaching the output. All four entries will take six clock cycles to process. This example indicates that the pipelined design significantly increases the frequency and throughput compared to the non-pipelined design.

3.4. Shared memory and resources

Since the SPOT design contains units that have a similar architecture, it will be useful to share the same architecture, possibly adding a few adjustments, rather than duplicating

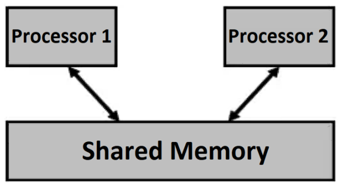


Figure 10: Shared memory strategy.

them for each unit. These adjustments will greatly optimize the use of area resources on the FPGA. In addition, shared resources can reduce complexity, increase productivity, reduce maintenance costs, and use the additional resources available for additional functionality. Provided that the shared architecture is free from defects or problems affecting the reliability, safety or security of the design. An additional step used to reduce area usage is memory sharing. In FPGA, shared memory refers to block and distributed RAMs accessible, in a clock cycle, by several different processing units within a parallel computation, which allows a very fast and efficient implementation.

3.5. Reliability

SRAM-based FPGAs are highly susceptible to the ionizing radiation environment in space, typically in the event of a nuclear explosion, and generate photocurrents through all of the semiconductor material, causing memory cells to switch and transistors to change the logical state randomly.

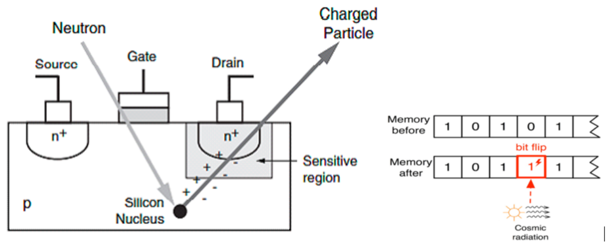


Figure 11: Radiation effect over SRAM.

A Single Event Upset (SEU) typically occurs by the change of logical state of a memory cell under the effect of a charged particle. It is a transient effect that will be erased by rewriting the affected memory cell. Any electronic circuit which has memory cells is likely to experience SEUs. This erroneous signal may remain in the digital system and can even propagate to other digital modules resulting in a failure. To mitigate space radiation effects in both their configuration and user memory, Triple modular redundancy and algorithms for error detection and correction are applied to reduce the susceptibility of the implemented SPOT algorithms to space radiations, increasing the system's reliability.

For best results, must manually design Triple-Modular Redundancy (TMR), for its part, involves implementing three instantiations of the Processor Element (PE) with

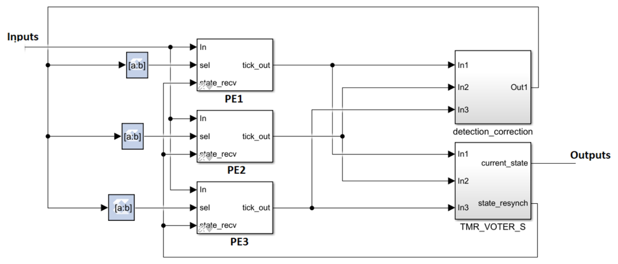


Figure 12: Triple Modular Redundancy with error detection and correction.

the majority voting upon the outputs. Typically, a TMR implementation will require spatial separation of the logic within the FPGA to ensure that a SEU does not corrupt more than one of the three instantiations. Another way to mitigate the effect of SEU is to exploit the Soft Error Mitigation (SEM) Intellectual Property (IP) cores provided by Xilinx to perform SEU detection, correction and classification for configuration memory. The cores utilize device primitives such as Internal Configuration Access Port (ICAP) and FRAME_ECC (Error Correction Code) to the clock and observe the Readback CRC (Cyclic Redundancy Check) feature to continuously scan the configuration cells. For SEU correction, the IP cores perform the necessary operations to locate and correct errors.

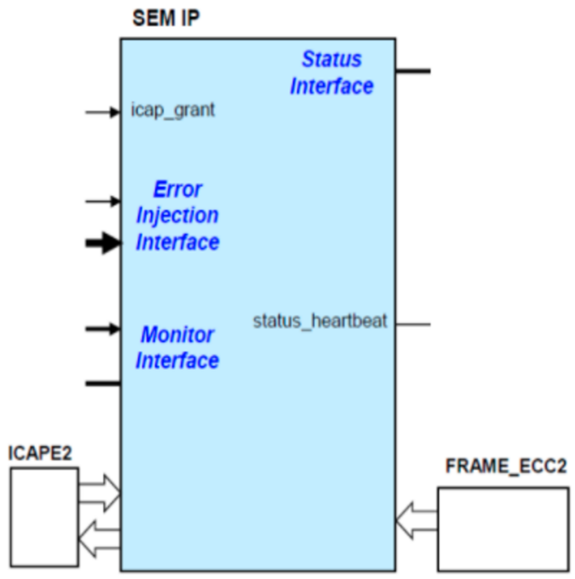


Figure 13: Software error mitigation IP.

The SEM IP cores also perform emulation of SEUs by injecting errors into configuration memory. The error injection feature provides a means to evaluate and test the SEU mitigation capabilities of the IP cores without the need for expensive test time at a radiation effects facility.

4. Input Images Acquisition Campaign

Images of the testing activities come from a acquisition campaign our team carried out using a Nikon D-3100 camera paired with an 18-105mm Nikkor optical system in Campo Imperatore, Gran Sasso, Italy. The acquisition system was mounted on a tripod and mounted with the Earth rotating and pointing toward any part of the sky interested in one or more satellites. Hundreds of 960 x 640 monochrome images were collected with 17 known and 15 unknown objects still to be identified. Exposure times are set to 5 seconds with a 2-second pause. The target of this expedition was the LEO satellites, the only satellites our acquisition systems could detect. More details in terms of field of view and focal length can be found in Table 1.

TABLE 1: Experimental Set-up info

Features	Values
Location	Campo Imperatore, Gran Sasso, Italy
GPS Latitude	42° 26’ 11.088” N
GPS Longitude	13° 36’ 30.018” E
GPS Elevation	1710 m
Camera	Nikon D3100
Optical System	Nikkor 18-105 mm
Platform	Tripod
focal length	24 mm
Size	960 x 640 px (Resized)
Exposure Time	5 sec
Pause	2 sec
Vertical FOV	35.57°
Horizontal FOV	51.40°

A sample from this campaign is shown in Figure 14. As can be seen, the two brightest groups of streaks in the sequence are LEO satellites (brightest one is the International Space Station while the other one is a Starlink).

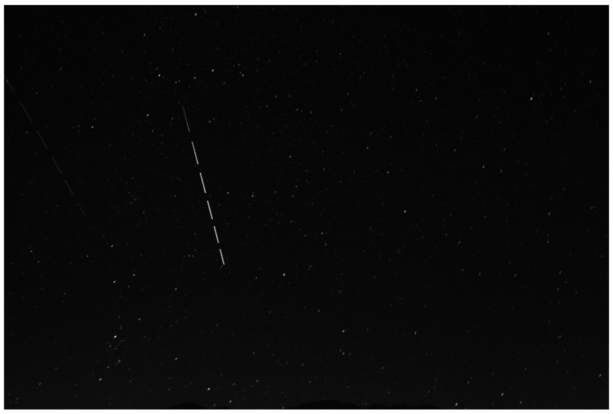


Figure 14: Sample images superposition from acquisition campaign. The brightest object is the International Space Station.

5. Test and Results

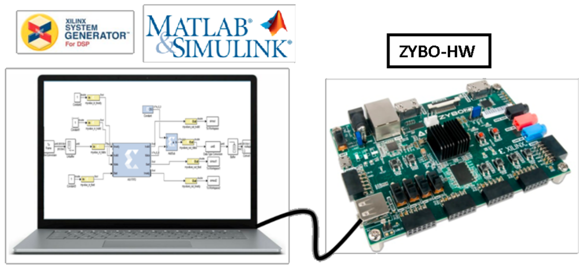


Figure 15: HIL set up.

In this section: unit test campaign, HIL set-up and results are shown and discussed.

Test campaign is organized in a stack of tests where on board modules are implemented and tested:

1) Pre-Processing Unit Test
2) Clustering Unit Test
3) Cluster Fusion Unit Test
4) Antitracking Unit Test
5) Cluster Growth Unit Test

5.1. HIL setup description & Validation methodology

A Hardware in the loop setup is developed as well (Fig. 15), using System Generator, to verify the performance and robustness of the control algorithms and simulating critical scenarios. System Generator includes a HIL-simulation tool that can be used for simulating together with FPGAs. The FPGA is connected to a computer by JTAG cable and inputs-vectors are simulated on the computer(Simulink) and sent to the FPGA. The FPGA performes the calculation then it produces the outputs to be sent back to the computer (Simulink).

Then, the outputs will be compared with the ones from a MATLAB copy of the target code. The comparison of the implemented codes with the reference MATLAB ones will be used to validate the successful implementation of the tested item. The workflow used for every target code is the following steps:

- Inputs are taken from their specific folder
- Run the MATLAB copy of the code
- Collect and store the outputs
- Run the Implemented code using the same Inputs
- Collect and store the outputs
- Board and MATLAB outputs are compared and discussed

A test campaign is also considered in order to generate input for the next executing unit at each step.

5.2. Results

5.2.1. Pre-Processing Unit Test. In this test, six acquisition campaign images were considered and Pre-Processing module was applied for each of them. This was carried out both with FPGA and MATLAB codes and then results were compared. Results of Pre-Processing unit are segments and energies related to the over thresholds pixels of the image. This is more convenient in order to store less information rather than having a mask with all the active pixels' energy values. An example of input and related reconstructed output is shown in Fig. 16 just for the most relevant part of the image.

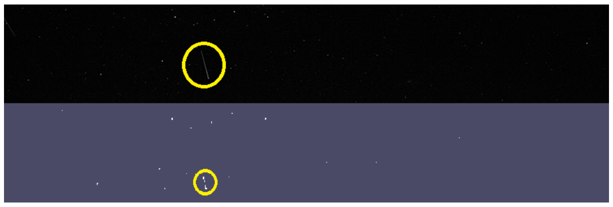


Figure 16: Detail of Pre-Processing module input (top) and output (bottom).

To compare results of both MATLAB and FPGA module we used performance indices described in [7]. These indices are defined to assess the similarity between two masks produced by different segmentation algorithms. These masks are matrices of binary numbers. When they are summed element by element, they give a two dimensional array of 0,1 and 2 values. By recording the number of 0 elements ($cntr_0$), 1 elements ($cntr_1$) and 2 ones ($cntr_2$) it is possible to compute two indices: G and G_0. The third index is defined as the ratio between the number of active pixel in mask 1 ($cntr_{m1}$) and mask 2 ($cntr_{m2}$). Evaluating each index for each image, the following value are obtained:

- $G = \frac{cntr_0 + cntr_2}{cntr_0 + cntr_1 + cntr_2} \times 100 = 100\,\%$
- $G_0 = \frac{cntr_2}{cntr_1 + cntr_2} \times 100 = 100\,\%$
- $M = \frac{cntr_{m1}}{cntr_{m2}} = 1$

Their values mean that there is a perfect superposition between MATLAB and FPGA codes' output masks for what concerns the localization of over threshold and under threshold pixels ($cntr_1 = 0$). Moreover, another comparison has been done also with segments' energies and it shows a complete success of the FPGA Pre-Processing Unit implementation.

5.2.2. Clustering Unit Test. In the implementation of the clustering unit, the same pre-processing test image was considered and its outputs were injected to the clustering unit.

Segments' positions, energies, weighted energies, and lengths for the the clustering unit are provided to both MATLAB and FPGA. The output data are the coordinates, energies, and dimensions of the Clustering. To compare the output data, the difference (ϵ) between the output data from different sources (y_{FPGA} and y_{MATLAB}) was evaluated:

$$\epsilon = y_{FPGA} - y_{MATLAB} \tag{5}$$

In Figures 17-20 these differences are related to the Primitive Clustering Part. Here the blue color is related to the difference distribution while the red color is the mean value, and the black color is used for the $\pm\sigma$ curves. "Col" and "Row" represent the y and z centroids' coordinates in the image plane reference frame, respectively.

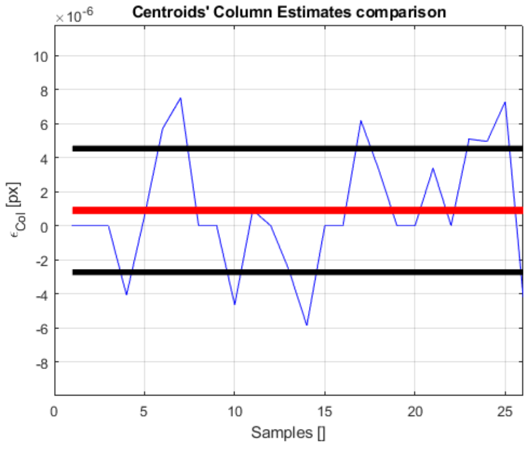


Figure 17: Cluster centroids' columns estimate differences over the whole samples set.

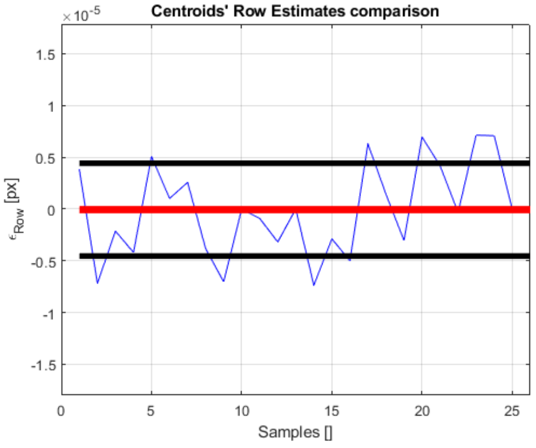


Figure 18: Cluster centroids' rows estimate differences over the whole samples set.

The centroids column and row plots show a predicted trend with an average value of zero, considering the interval $\pm 3\sigma$ the differences are still very far from the maximum tolerance of 10^{-1}. When comparing clusters' dimension and energies, the output data from FPGA and MATLAB are matched due to not applying truncation and rounding, as result of their integer type values unlike the centroids estimate.

To see the effects of the Improved clustering module, Fig. 21 is considered.

Here in the first two columns from left are the coordinates of the segments that compose primitive clusters. As

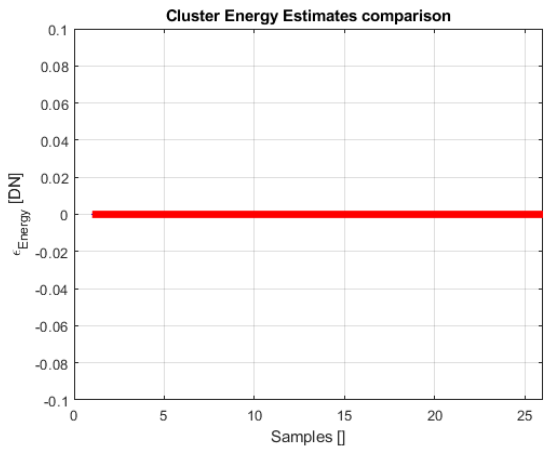


Figure 19: Cluster centroids' energies estimate differences over the whole samples set.

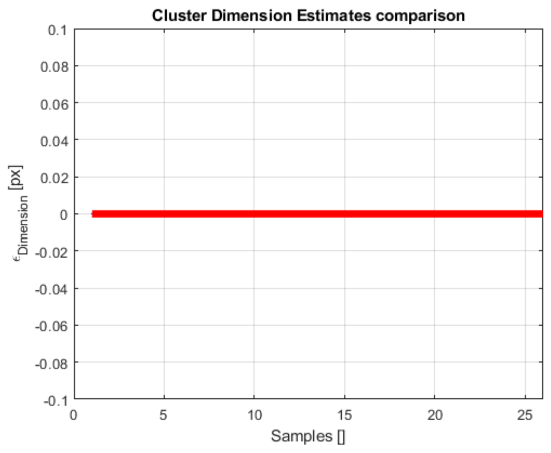


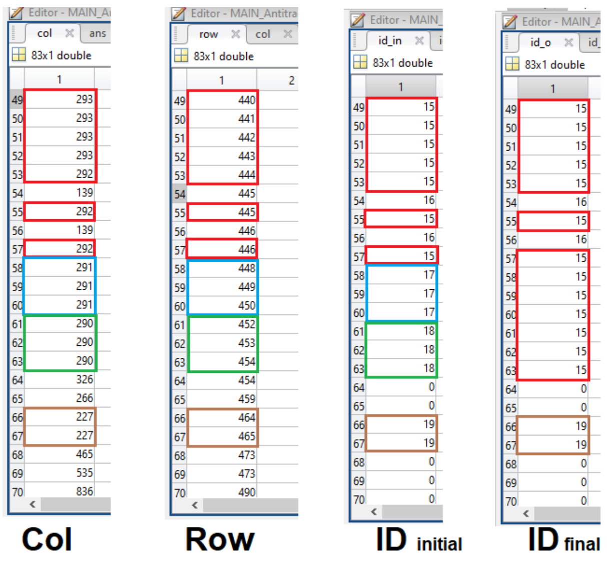
Figure 20: Cluster centroids' dimensions estimate differences over the whole samples set.

can be seen on the last two columns on the right, a change in their corresponding clusters identifier (ID) appears.

Before improved clustering, each segment has an associated cluster identifier.

As a result of applying several filters, and identifier re-assigning, the primitive clusters ID=17 and ID=18 are grouped with clusters ID=15. Hence, a large cluster of ID=15 was formed (red boxes). The other clusters which are not merged still preserve their original index as can be seen for cluster ID=19 (brown box). The three primitive clusters 15,17,18 which are merged in the end are the streak fragments associated with the ISS cluster Fig. 22.

5.2.3. Cluster Fusion Unit Test. In the cluster fusion module test, two consecutive images input were provided to the target unit. Clusters of the first and second images are compared using the previously described filters in the architecture section and here the considered images are merged and displayed (Fig. 23). The results provided by



	Col	Row	ID initial	ID final
49	293	440	15	15
50	293	441	15	15
51	293	442	15	15
52	293	443	15	15
53	292	444	15	15
54	139	445	16	16
55	292	445	15	15
56	139	446	16	16
57	292	446	15	15
58	291	448	17	15
59	291	449	17	15
60	291	450	17	15
61	290	452	18	15
62	290	453	18	15
63	290	454	18	15
64	326	454	0	0
65	266	459	0	0
66	227	464	19	19
67	227	465	19	19
68	465	473	0	0
69	535	473	0	0
70	836	490	0	0

Figure 21: Improved clustering effects on cluster identifier evolution.

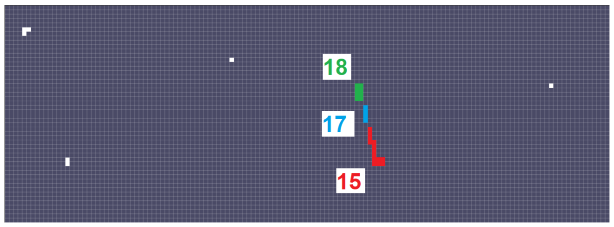


Figure 22: ISS' streak primitive clusters.

FPGA have been compared with MATLAB and they both show the same results: five clusters had been correctly merged. Four of them are related to stars while the remaining one is the ISS. In particular, Fig. 24 shows centroids relative to the ISS' RSO in the first and second images of the considered couple, along with the final structure resulting from the fusion operation.

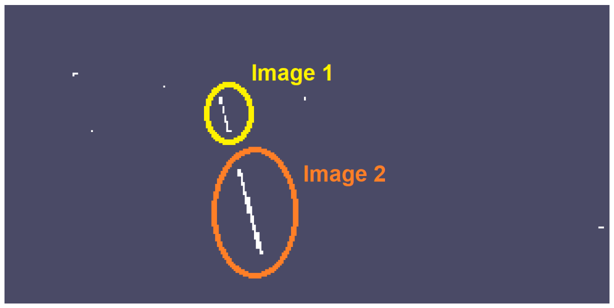


Figure 23: ISS' fused streaks by Cluster fusion module.

5.2.4. Antitracking Unit Test. For the anti-tracking test, a single image with its associated sensor position was con-

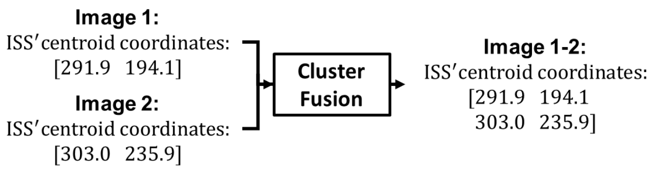


Figure 24: ISS' fused centroids by Cluster fusion module.

sidered. The fixed position of the camera with the ground and the short interval of shots do not result in a significant variation of the camera attitude. The considered image is shown in 25.

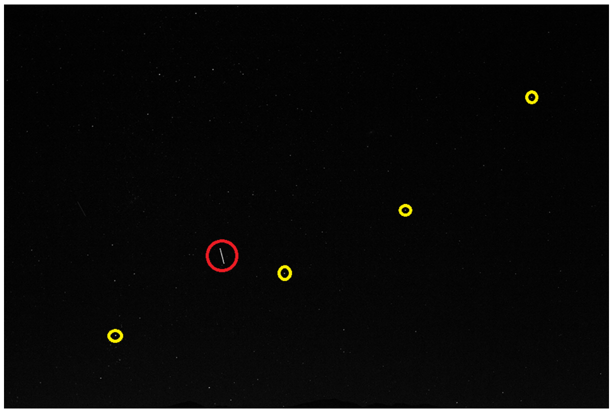


Figure 25: ISS (Red circle) and stars(Yellow circle)

In particular, as can be seen, five Cluster was detected, of which four stars and one object (ISS). After applying the antitracking, all the objects recognized as stars (yellow circles in the figure) are filtered out using the Hipparcos catalog, while the ISS is not deleted (red circle in the figure) and is included in the Antitracking outputs to be processed by the Cluster Growth unit.

To perform Antitracking, the following attitude quaternion was estimated and used:

$$q = [-0.526829; 0.222090; -0.342451; -0.745556] \quad (6)$$

Starting from the following objects' centroids, of which the first four rows are relative to the four detected stars and the last row is relative to the ISS' centroid:

$$centroids = \begin{bmatrix} 178.2 & 116.4 \\ 445.5 & 215.3 \\ 637.4 & 315.5 \\ 837.5 & 494.6 \\ 345.5 & 399.0 \end{bmatrix} \quad (7)$$

where, the first column represents the detected centroids' y-coordinate in the image plane and the second column represents the detected centroids' z-coordinate in the image plane, after the Antitracking module, clusters relative to stars are recognized as stars and deleted and only the cluster relative to the ISS remains:

$$centroids = \begin{bmatrix} 345.5 & 399.0 \end{bmatrix} \quad (8)$$

Here again, MATLAB and FPGA results are perfectly matched.

5.2.5. Cluster Growth Unit Test. For the last test, all the six consecutive images processed in three couples are used. The entire sequence is shown in Fig. 14. By Cluster Growth results, the unique object in the FOV was tracked (the ISS). Its centroid coordinates evolution is shown in Table 2. MATLAB and FPGA outputs are numerically the same, considering the first digit of the decimal part. This is because, by project analysis, the Cluster Growth centroids are rounded to the first digit.

TABLE 2: ISS centroids in the image sequence

Image	Column (px)	Row (px)
1	291.9	194.2
2	303.0	235.9
3	315.1	281.9
4	326.3	324.9
5	336.4	363.6
6	345.5	399.0

6. Conclusions

In this work, an overview of the SPOT mission, the proposed software architecture together with its implementation was presented. The implementation approach using System Generator made it more practical and faster in comparison to the other software. The techniques applied for the implementation show a significant effect in terms of area usage and the data throughput of the FPGA, resulting in a suitable solution for the use in real-time missions.

Hardware-in-the-loop simulations are performed to validate the SPOT algorithm and its implementation. A night sky image acquisition campaign has been performed for test input data collection. Preliminary results show that all the implemented algorithms were validated and the provided outputs are compliant with project requirements.

The aim of this work is to verify and validate the correctness of the implemented algorithm against the theoretical results. In the future more tests will be applied to evaluate the proposed implementation approach in terms of cost-performance ratio. In addition, several integrations will be carried out, such as the implementation of State Machine on the Microprocessor part of the SoC for controlling data flow and decision making. These will include the implementation of a CAN interface to connect it to the on-board camera as well as other communication protocols between the on-board computer and the host platform to prepare the SPOT software/hardware payload for its IOV mission.

References

[1] H. Klinkrad, T. Donath, and T. Schildknecht, "Investigations of the feasibility of a european space surveillance system," in *Proceedings of the 7th US/Russian Space Surveillance Workshop, Monterey, California*, vol. 29, 2007.

[2] T. Flohrer and H. Krag, "Space surveillance and tracking in esa's ssa programme," in *Proceedings 7th European Conference on Space Debris, Darmstadt, Germany, https://conference. sdo. esoc. esa. int*, vol. 1, 2017.

[3] J. Utzmann and A. Wagner, "Sbss demonstrator: A space-based telescope for space surveillance and tracking."

[4] V. Abbasi, S. Thorsteinson, D. Balam, J. Rowe, D. Laurin, L. Scott, and M. Doyon, "The neossat experience: 5 years in the life of canada's space surveillance telescope," in *1st NEO and Debris Detection Conference*, vol. 22, 2019.

[5] D. Spiller and F. Curti, "A geometrical approach for the angular velocity determination using a star sensor," *Acta Astronautica*, 2020.

[6] D. Spiller, E. Magionami, V. Schiattarella, F. Curti, C. Facchinetti, L. Ansalone, and A. Tuozzi, "On-orbit recognition of resident space objects by using star trackers," *Acta Astronautica*, vol. 177, pp. 478–496, 2020.

[7] M. Mastrofini, F. Latorre, I. Agostinelli, and F. Curti, "A convolutional neural network approach to star sensors image processing algorithms."