Clustering Algorithm for a spaceborne Lightning Imager: design, trade-off and FPGA implementation.

Pietro Nannipieri, Daniele Davalle, Stefano Nencioni, Paolo Lombardi and Luca Fanucci

***Abstract*—In this paper, the development and implementation of the Clustering Algorithm developed for an instrument to detect lightning phenomena (Lightning Imager) is presented. The aim of the Lightning Imager instrument is to provide information relevant to the localization and the radiance of lightning events with respect to terrestrial systems. A CMOS sensor with 1170×1000 resolution and 1000 fps frame rate is used on the instrument, making the Lightning Imager able to identify the smaller lightning in terms of dimension (minimum diameter 10 km) and temporal pulse duration (minimum 0,6ms). The amount of acquired data is very large but only a small subset of pixels contains useful lightning information, thus information clustering is fundamental for downlink data-rate reduction. Raw sensor data are processed by an ASIC in order to extract pixels coordinates belonging to lightning flashes (Detected Transients) which are then directly sent to the Clustering Algorithm implemented in an FPGA. Detected Transients must be processed as they come to the Clustering Algorithm due to high throughput requirement. The Clustering Algorithm objective is to define rectangular windows that enclose more Detected Transients in order to send more compact information to Earth. The design presented in this paper was carried out through the development of an high-level model in order to verify algorithm functionality and compliance with performance requirements. A lightning event generator was developed to simulate the Detected Transients coming from the ASIC part.The hardware architecture was conceived and a bit-true model was developed to evaluate the implementation loss.**

***Index Terms*—Lightning Imager, Clustering Algorithm, RTAX2000.**

I. Introduction

Clustering is a very general problem in which many researches have been carried out, within a broad range of fields of study. Its applications can be found in the fields of data mining, image analysis, pattern recognition to name just a few. Basically, it consists in classifying a set of unlabelled objects into groups of "similar" objects. The concept of similarity is defined by the specific application and the algorithm used to solve the clustering problem. This paper presents a clustering algorithm for a space borne Lightning Imager: In section I a brief literature review of the clustering algorithm is done, and the system where the clustering algorithm will be implemented is described at block and requirement level; in section II a data saving preliminary estimation achievable adopting a clustering algorithm on the described application is derived; in section III the detailed description of the adopted clustering algorithm is given, together with its implemented hardware architecture; in section IV results are presented in terms of execution time and hardware complexity; finally in section V conclusions are drawn and possible future development of the work are proposed.

P. Nannipieri and L. Fanucci are with the Department of Information Engineering, University of Pisa, Italy (e-mail: pietro.nannipieri@ing.unipi.it luca.fanucci@unipi.it).

D. Davalle is with IngeniArs S.r.l., Pisa, Italy (e-mail: daniele.davalle@ingeniars.com).

S. Nencioni and P. Lombardi are with Leonardo S.p.A., Campi Bisenzio, Firenze, Italy (e-mail: stefano.nencioni@leonardocompany.com paolo.lombardi@leonardocompany.com).

A. Clustering algorithms

A review on clustering techniques is presented in [1], where a classification of the different approaches used in clustering is described. Clustering algorithms can be divided in incremental and non-incremental algorithms. Non-incremental algorithms assume to have the whole distribution of elements to be clustered, while incremental algorithms process elements one by one. Non-incremental algorithms include the k-means and density-based algorithms as DBSCAN [2]. K-means algorithm has known drawbacks, i.e., the number of clusters must be known *a priori* and it aims at identifying spherical clusters, not arbitrary shapes. Multiple scans of data are also required, therefore k-means is not directly applicable to the streaming approach [3]. Algorithms based on density are useful to recognize data with arbitrary shapes. Density-based algorithm also overcome the problem of the number of clusters, which can be unknown. Incremental clustering algorithms belong to the category of streaming algorithms, i.e., algorithms for processing data streams. These algorithms process data point-by-point, without considering the whole image. Incremental algorithms are useful when memory resources are limited and the execution time is an issue. Points are processed in a single scan without being stored and randomly accessed. STREAM [4] is based on the k-median algorithm for streaming data. The data is divided into blocks and locally clustered, then all the local medians are clustered to produce the final medians. CluStream [5] uses online and offline components for the clustering. The online component collects information of the streaming data into micro-clusters. Micro-clusters are finally clustered with k-means by the offline component. Clustream often produces spherical clusters because it takes distance as measurement. D-Stream is a streaming algorithm based on density and grids [3]. The algorithm uses a two-phase approach, inspired by CluStream. In the online phase data is recorded into a grid, the density vector is updated, then in the off-line phase clusters are adjusted once every number of

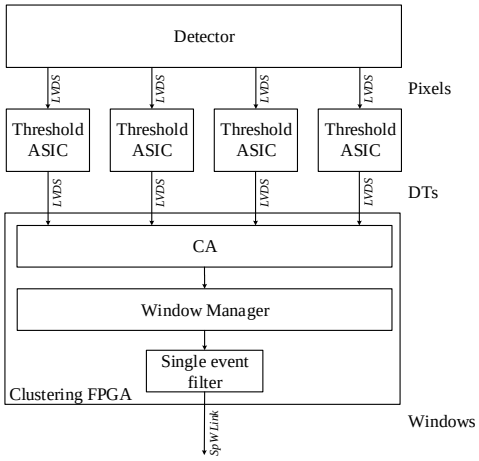

Fig. 1. System block diagram

predefined steps. The grid is useful for a first classification of raw data, to avoid raw data storing. The algorithm adjusts clusters over time, thanks to a density decaying technique. D-Stream is concerned on dealing with high-dimensionality data and the evolution of clusters over time. MD-Stream is an extension of D-Stream to cope with variable density clusters [6]. DenStream [7] is a streaming algorithm based on DBSCAN. [8] presents a streaming version of the fuzzy c-means algorithm: the online fuzzy c-means. Data is processed in chunks, each one processed with fuzzy c-means. The results coming from each chunk of data is merged into the final result.

B. Lightning Imager System Description

A possible use case for the implementation of the clustering algorithm is presented in this section. A Lightning Imager (LI) instrument is a system, generally mounted as satellite payload, able to detected and map the presence of lighting on various geographical location. To do that, the system needs to process a considerable amount of data in a restricted time interval, considered the natural characteristics of the lighting phenomena. An example of a LI instrument is presented in [9], together with an architectural system description including the optical part.

The block diagram of the LI processing system considered for this work is shown in Figure 1. Pixels are acquired by a 1170×1000 detector. The sensor frame-rate is 1000 fps which means that the amount of acquired data is very large, thus information clustering is required to reduce the downlink data-rate. First, these pixels must be processed to detect the lightning events. Pixels are sent directly to the a dedicated "Threshold Application-Specific Integrated Circuits (ASICs)" trough dedicated Low Voltage Differential Signallings (LVDSs) signal lines. The "Threshold ASICs" separately elaborate pixels coming from the four quadrant to estimate the background and to identify Detected Transients (DTs)s. In fact the surface of the Earth is not uniformly illuminated, thus a background-estimation is carried out and an adaptive-threshold is used. Pixels whose difference with the estimated background is above the threshold are defined DTs and are considered as lightnings. Threshold is set as a trade-off between false-alarm and detection efficiency and is assumed to be the same over all the Field Of View (FOV), therefore the detection probability is constant for all the pixels while the false-alarm rate is not. DTs are fetched by the Field-Programmable Gate Array (FPGA) trough dedicated LVDSs connection; the extracted DTs are processed by the Clustering Algorithm (CA) module, which must be capable of processing 1000 events in 1 ms. Finally, clustered and filtered data are temporary stored in output buffers, ready to be transmitted to the Host (i.e. a mass memory) trough the SpaceWires (SPWs) interface of the system. The frame is divided into 4 quadrants for parallel processing of 4 different sections of the sensor. In this work, both the quadrant-specific clustering and full-frame clustering were treated. In quadrant-specific clustering, the DTs coming from the different quadrants are processed by separate CA units, to exploit hardware (HW) parallelisation. Full-frame clustering foresees that all the DTs are processed by a single CA unit. This technique may be useful because one of the system requirement is that 813 of the 1000 DTs can be concentrated into one single quadrant. This high concentration is therefore voiding the benefit given by quadrant parallelization, because each quadrant CA unit must be almost as powerful as the full frame CA unit. Given the nature of lightning flashes seen from above the clouds, the maximum window size is 8×8. CA modules produce the clustering windows for each quadrant. Occasionally some spurious DTs can be present depending on the noise level affecting the sensor. If spurious DTs are isolated, i.e., far from other DTs relative to real lightning events, they can be recognized by observing the radiometric data of their 8-pixel neighbourhood. To this aim, a 3×3 window filter is present in the system: every 3×3 window produced by the CA is analysed and possibly discarded. This technique improves noise robustness and reduces the volume of data to be transmitted, as described in Section II.

System optimisation has a fundamental role in the reduction of the data rate. From the point of view of the algorithm, compare operations are limited to the necessary ones. In window-window comparisons, only windows near the DT are selected for comparison with the window candidate for DT inclusion. In this way, only windows which can really interfere with window expansion are taken into consideration, avoiding processing time waste. Other compare operations are avoided by considering the DT ordering, i.e., the usual raster scan. From the point of view of HW architecture, with the aim of reducing the DT processing time, data loading and processing is parallelized as much as possible. The limit to parallel processing is represented by the area occupation. The target device for the LI CA is a Microsemi RTAX2000, a radiation-tolerant FPGA from Microsemi with 30 k to 500 k ASIC gates (250 k to 4 M system gates), largely employed in several space missions [10]. The work presented in this paper describes the design and implementation process of the clustering algorithm, which in future may be integrated with the overall system containing also the detector and the "threshold ASIC".

II. Clustering rationale

In the following, a simplified calculation of data saving is presented to justify the clustering strategy.

Without noise filtering and clustering, all DTs are packeted and transferred independently. Each packet is composed of a 3-byte packet header, 3-byte DT coordinates, radiometric (12 bit per pixel) and background (16 bit per pixel) data of a 3x3 window around the DT. To understand why the size of the considered window is 3x3, note that a single pixel is equivalent to an area of 10×10 km^2 and that the average dimension of a lighting event is roughly 30×30 km^2, as indicated in a similar work [11]. The total data to be transferred for each DT is $S_{DT} = (3+3) \cdot 8 + [(12+16) \cdot (3 \cdot 3)] = 300$ bits. Suppose to have 1000 DTs per millisecond, i.e., $N_{DT} = 1000$. Considering that in a communication protocol such as SpaceWire (SpW) [12] 2 bits have to be added every 8 bits and for each packet there has to be a 4 bits end-of-packet, the amount of data to be transferred in Mbps is:

$$R_{TX} = \frac{N_{DT}}{1000}\left[\frac{10}{8}S_{DT} + 4\right] = 379\,\text{Mbps} \qquad (1)$$

If noise filtering only is activated, assuming that $N_{DTi} = 250$ of the 1000 DTs are isolated events and that the filter has an efficiency $\eta = 87\,\%$, $250 \cdot \eta$ noise events are filtered out. The amount of data to be transferred in this case is:

$$R_{TX} = \frac{N_{DT} - \eta N_{DTi}}{1000}\left[\frac{10}{8}S_{DT} + 4\right] = 297\,\text{Mbps} \qquad (2)$$

leading to a saving of 22 %.

If clustering only is activated, assume that the 750 aggregated DTs are clustered under half-full 7×7 windows, which means that each window has half of its pixel by an event. Therefore, we have $N_{W7\times7} = \left\lceil \frac{750}{7\cdot7/2} \right\rceil = 31$ 7×7 windows. The packet used for transferring a 7×7 window is composed of the 3-byte header, the 3-byte coordinate of the top-left pixel plus 2 bytes for window width and window height, $\left\lceil \frac{12\cdot(7\cdot7)}{8} \right\rceil = 74$ bytes for radiometric data and $\frac{16\cdot7\cdot7}{8} = 98$ bytes of background information. The total packet size to transfer a 7×7 window is $S_{W7\times7} = (3+3+2+74+98 = 1440$ bits. The amount of data to be transferred in this case is:

$$R_{TX} = \frac{N_{W7\times7}}{1000}\left[\frac{10}{8}S_{W7\times7} + 4\right] + \\ + \frac{N_{DTi}}{1000}\left[\frac{10}{8}S_{DT} + 4\right] = 159\,\text{Mbps} \qquad (3)$$

leading to a saving of 60 %.

If both noise filtering and clustering are activated, in the same conditions, the amount of data to be transferred is:

$$R_{TX} = \frac{N_{W7\times7}}{1000}\left[\frac{10}{8}S_{W7\times7} + 4\right] + \\ + \frac{(1-\eta)\cdot N_{DTi}}{1000}\left[\frac{10}{8}S_{DT} + 4\right] = 71\,\text{Mbps} \qquad (4)$$

leading to a combined saving of 82 %.

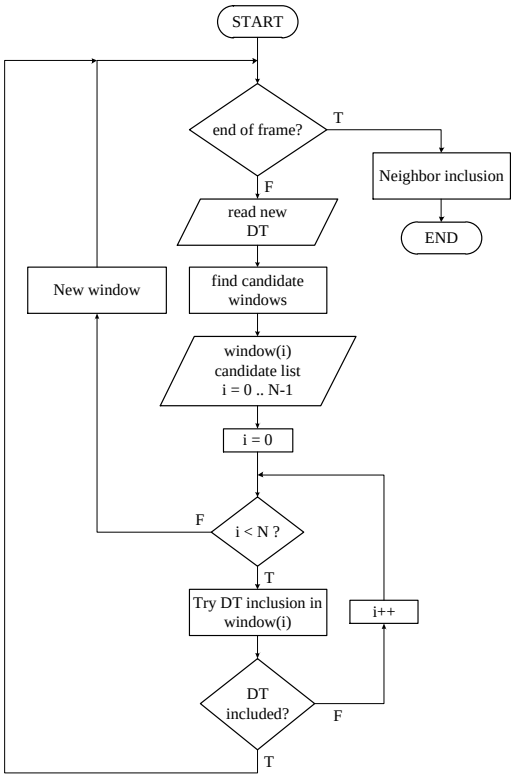

Fig. 2. High level algorithm flowchart

III. Clustering algorithm

As seen in Section I-B, LI adopts the clustering technique in order to save data to be transmitted to the ground station, as described in Section II. Performance requirements are stringent and the clustering must be achieved within a frame period.

Considering that the interface data throughput is limited, a low-level clustering is only possible. Further processing is delegated to ground.

With the aim of the hardware implementation, the algorithm and the HW architecture were jointly optimized to reach the desired performance within the limits of the selected device.

With reference to the classification reported in Section I-A, the clustering algorithm used in LI falls under the category of the incremental clustering algorithms, since input data arrives in a streaming form. Non-incremental approaches such as k-means or DBSCAN are not feasible since incoming DTs cannot be stored in internal memory and the throughput requirement impedes the usage of these algorithms. The algorithm is bidimensional, input DTs are represented on $(x,\ y)$ rectangular coordinates.

LI CA is non-iterative, i.e., as DTs are clustered, the clustering configuration is not further refined as in partitional or hierarchical algorithms. Only a single-pass approach is feasible, considering the short processing time available and the number of DTs to be clustered every frame. Two-phase solutions like CluStream and D-Stream were therefore discarded.

The number of clusters is not known *a priori*. However, the maximum expected number of clusters is derived on a statistical basis. Therefore algorithms as STREAM, fuzzy c-

means in which the number of clusters is a parameter were not considered.

LI CA defines rectangular windows containing all DTs. The size of such windows is dynamically determined (see algorithm flowchart in Figure 2) based on if there are still adjacent DTs that can be embedded. This choice allows easy manipulation of windows, calculations for DT adjacency and window overlap checks are simpler to carry out with respect to other types of windows, e.g., circular windows. The adoption of rectangular windows, together with the usage of a simple distance metric allow an efficient hardware implementation. The algorithm computation requires fixed point additions and comparisons only.

Additionally, rectangular windows are fully characterized with four parameters, therefore they are easily manageable and memory-efficient when compared to more complex shapes. The set of window parameters adopted in this work is the vector: $(x,\ y,\ \Delta x,\ \Delta y)$, where $x,\ y$ are the coordinates of the top-left corner of the window, while $\Delta x,\ \Delta y$ are the width and height of the window, respectively.

Background information and radiometric data are transferred for every pixel in each window. In order to reduce the transmission of insignificant data, windows must contain at least one DT for each row and column. Similarly, complete window overlapping has to be avoided, since duplicate information is transmitted to ground if overlapping is allowed.

Almost all the algorithms presented in Section I-A present a dynamic nature, in the sense that they cope with the evolution of data and clusters evolve to follow the distribution of points. LI CA initiates a new clustering operation every frame, in order to simplify the on-board processing.

Online window merging and splitting are not implemented by LI CA to reduce the processing. This drastically reduces the computation time but leads to a less optimal clustering, which, according to our analysis, is perfectly acceptable for this application.

Isolated DTs are included in a dedicated 3×3 window when the noise filter described in Section I-B is activated. A decision whereas the analysed windows contains valid information is made thanks to it: if only a single pixel is active within a 3×3 window, it is considered to be false (incompatible with the size of the natural event). Otherwise, the event is considered to be true, neglecting the very low probability of two adjacent false pixel.

A. Algorithm description

Figure 2 shows the frame elaboration flowchart of the clustering algorithm.

For each DT read from the input FIFO, the algorithm scans the actual defined windows. Windows neighbouring the new DT are "candidates" to be expanded to include the new DT. A window is considered neighbour to the new DT if the DT is located in the one-pixel frame around the window. In general, there are more than one candidate for each DT, therefore the output of the search is a list of windows. DT inclusion is attempted on the candidate windows according to predetermined priority rules. DT inclusion can fail because

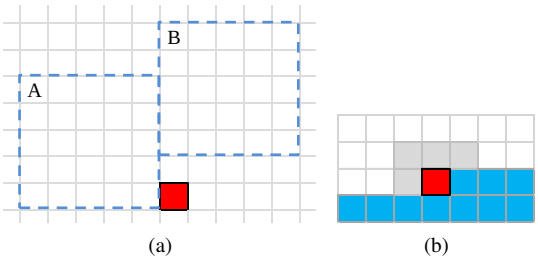

Fig. 3. (a) Window overlapping case. (b) The new DT, the neighbouring pixels to be explored for windows (grey) and future pixels (blue)

the candidate window has reached the maximum size or the candidate window expansion produces overlap with another window, as shown in Figure 3a.

If there are no candidate windows or neither one of the found candidates can be expanded to include the new DT, a new window is opened.

This process is repeated for each DT in the frame. At the end of the frame, the windows are transmitted and window neighbours are included according to the neighbour setting.

Neighbours can be deactivated, activated or selectively activated. When neighbours are deactivated, windows are transferred as they are. If neighbours are activated, a one-pixel frame is added to every window. If neighbours are selectively enabled, the one-pixel frame is added to 1×1 windows only.

Figure 3a shows an example where window overlapping could occur. The new DT is next to window A, which is the only candidate for expansion as it is the only adjacent one. If window A is expanded one pixel to the right to include the new DT, a portion of window A overlaps window B.

Candidate windows are searched for in the 8 pixels neighbouring the new-event. Given the raster ordering of the DTs, only 4 pixels are worth looking for neighbouring windows. Figure 3b shows the new event (red) and the 4 neighbouring pixels to be checked (grey). The other 4 neighbouring pixels (blue) are not to be explored because they still have to be processed and since no window neighbours are added while clustering, no windows are present in those blue pixels. Window neighbours (if required) are added when the frame

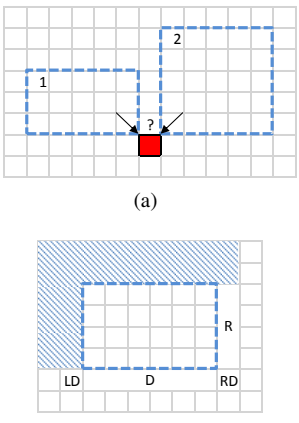

Fig. 4. (a) Window expansion conflict. (b) Allowed window expansion directions

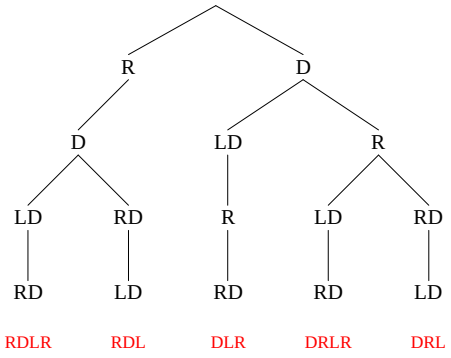

Fig. 5. Possible clustering strategies

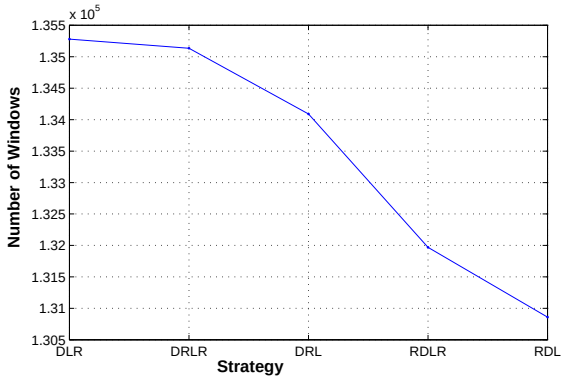

Fig. 7. Number of windows as a function of the strategy

is fully processed. Neighbour attachment while clustering of events was also considered (dynamical neighbour attachment). Solutions including dynamical neighbour attachment were discarded because they caused problems with total window overlap, single event windowing, and were also more computationally intensive. Neighbour attachment in post-processing avoids total window overlap, the maximum window overlap is 2 pixels wide. The actual data reduction provided by the clustering algorithm depends then on DTs geometrical disposition: purely horizontal/vertical events lead to bigger data reduction, while purely diagonal disposition of DTs could theoretically, for more than 4 diagonal DTs, lead to data growth. However due to the nature of the lightning phenomena, purely diagonal disposition of DTs has low probability, thus the eventual increase of data to be sent is not considered to be a problem for the system.

1) Clustering strategies: A strategy for the clustering is necessary to resolve situations such as the one depicted in Figure 4a. The strategy determines which direction is preferred for the expansion of windows. The strategy also impacts on the total number of windows defined. Therefore, the selection of the strategy is important for the minimization of the total number of windows.

Basically, since the DT ordering is the usual raster scan over the quadrant, it is not possible that a new DT is on the top edge of a window, i.e., windows cannot be expanded in the UP (U) direction. Figure 4b illustrates the possible expansion directions for windows. LEFT (L), LEFT-UP (LU), U, RIGHT-UP (RU) directions are not possible due to the raster scan.

RIGHT (R), RIGHT-DOWN (RD), DOWN (D) and LEFT-DOWN (LD) are the possible expansions for windows. A priority has to be defined with these directions. Referring to Figure 4a, if an R-before-L strategy was selected window 1 would have been expanded. On the other hand, with an L-before-R strategy, window 2 would have been expanded.

Figure 5 shows the strategy tree. The possible strategies are given a short name to be recalled in the following: RDL, DLR, DRL, RDLR, DRLR, according to the direction priority.

Figure 6 shows the order of expansion as a function of the position of the new event with respect to the window for the different strategies. The strategies are equivalent from the point of view of HW complexity. The best strategy is evaluated by means of computer simulation over a significant set of test-vectors.

Figure 7 shows the results of computer simulations over 1000 frames, with flashes concentrated in a 50×50 area. The higher the concentration of lightning flashes, the higher the impact of the chosen strategy, because a larger number of window expansion conflicts are expected. RDL is the best strategy from the point of view of window minimization, therefore it was chosen for the implementation. The adoption of the RDL strategy entails a saving on 1000 frames of:

- 243 windows with 190x190 cluster concentration
- 730 windows with 100x100 cluster concentration
- 4420 windows with 50x50 cluster concentration

2) Test-vector generation: The random test-vectors used for the clustering algorithm design and verification were generated with a MATLAB simulator.

In order to simulate the randomness of a lightning flash seen from above the clouds, the MATLAB model generates random points representing the flash according to a bi-dimensional Gaussian distribution.

First of all, the X and Y coordinates of the points are generated by means of two independent Gaussian distributions. The mean value along X and Y represents the flash centroid, uniformly distributed in the lightning concentration area. The standard deviation σ is correlated with the radius of the flash. In a Gaussian distribution, more than 99.7 % cases fall in

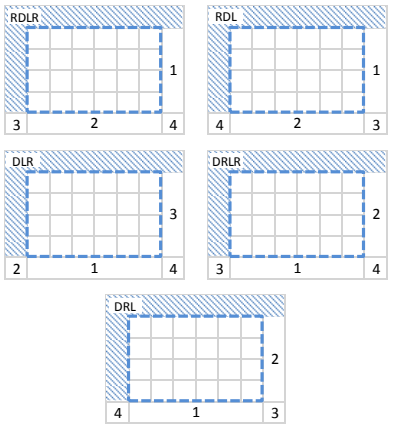

Fig. 6. Expansion direction priorities for the different strategies

the interval $[-3\sigma, 3\sigma]$. Therefore, 3σ can be considered as the radius of the distribution and we can obtain the standard deviation for the generation of flashes as: $\sigma = \frac{R}{3}$.

Then, the Y coordinates of the points are shrunk by the factor $(1-e)$, where e represents the distribution eccentricity. The eccentricity is modeled as a Gaussian random variable with selectable mean and variance.

Finally, the flashes are rotated by an angle θ, modeled as a uniformly distributed random variable in the interval $[0, 2\pi)$.

The generated coordinates of the points of the flashes are then quantized to integer numbers representing the pixel coordinates, considering that each pixel is about a 10×10 km^2 square.

A portion of the resulting generated test-vector is represented in Figure 8. The different strategic all have the same computational cost. The number of the necessary comparison and sums depends on the disposition of DTs.

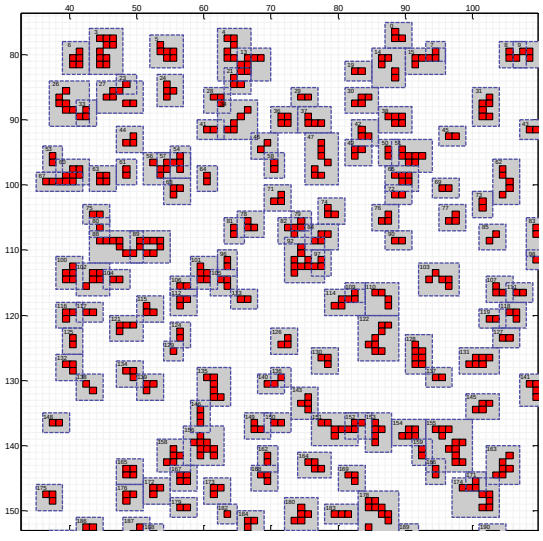

Fig. 8. Generated test-vector with windows evaluated by the clustering algorithm

B. Architecture description

The HW architecture of the CA is depicted in Figure 9.

The Window Memory is used to store the windows defined during the clustering process. The Window Memory Controller reads and writes windows in parallel. P_{AC} windows are accessed at the same time in order to allow the parallel processing of windows.

When a new DT is fetched, all the already defined windows must be accessed to look for candidates for DT inclusion. This operation is carried out by the Adjacency Check (AC) block. While windows are scanned for AC, they are also analysed for proximity with respect to the current DT by the Near-Window Check (NWC) block. A Near Window (NW) is defined as a window that could interfere with the expansion of the Candidate Window (CW). The results of the AC are stored in the CW registers, while NWC results are stored in the NW registers. Candidate Windows are 4 at most, while Near Windows are 8 in the theoretical worst case as it will be

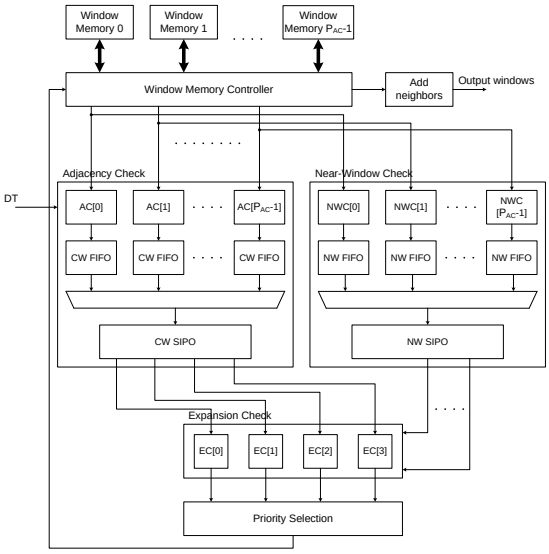

Fig. 9. LI CA architecture

proven in the following sections. NWC reduces the number of window-to-window comparisons in the Expansion Check (EC) phase. The EC block takes the candidates from the CW registers and checks their expandability with respect to all the NWs identified by the NWC. If there are more than one possibilities to expand the CW, the Priority Selection block selects the preferred expansion direction according to the RDL strategy, which grants the minimum number of windows, as described in Section III-A1. The architecture parallelism is parametric and can be adapted to push the area vs performance trade-off. In particular, the number of windows elaborated in parallel P_{AC} is a parameter. P_{AC} sets the number of AC and NWC units, since the processing of every window requires one AC and one NWC.

The number of EC units is 4, to be able to process the worst-case number of CW in parallel. The latency of the EC units is a parameter, and therefore the number of NWs that the EC units is able to process in parallel is selectable.

In the following sections, the most relevant blocks are detailed.

1) Adjacency Check: The AC block checks if the window being processed is adjacent to the current DT and outputs the direction of expansion for DT inclusion. The DT can either be:

- already internal to the window, so the window does not have to be expanded to include the DT, if $X_{WIN} \leq X_{DT} < X_{WIN} + \Delta X_{WIN}$ and $Y_{WIN} \leq Y_{DT} < Y_{WIN} + \Delta Y_{WIN}$;
- adjacent to the right of the window, so the window has to be expanded in the R direction to include the DT, if $X_{DT} = X_{WIN} + \Delta X_{WIN}$ and $Y_{WIN} \leq Y_{DT} < Y_{WIN} + \Delta Y_{WIN}$;
- adjacent to the bottom-right corner of the window, so the window has to be expanded in the RD direction to include the DT, if $X_{DT} = X_{WIN} + \Delta X_{WIN}$ and $Y_{DT} = Y_{WIN} + \Delta Y_{WIN}$;
- adjacent to the bottom of the window, so the window

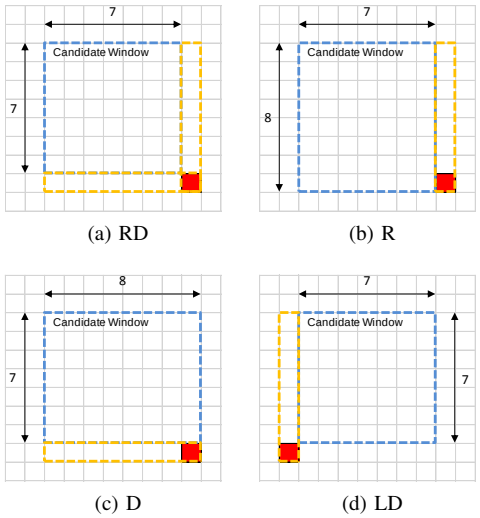

Fig. 10. The four cases of expansion of a candidate window. The blue dashed rectangle is the candidate window. Orange rectangles represent the area that must be free from NW to expand the CW in the given direction.

has to be expanded in the D direction to include the DT, if $X_{WIN} \leq X_{DT} < X_{WIN} + \Delta X_{WIN}$ and $Y_{DT} = Y_{WIN} + \Delta Y_{WIN}$;

- adjacent to the bottom-left corner of the window, so the window has to be expanded in the LD direction to include the DT, if $X_{DT} = X_{WIN} - 1$ and $Y_{DT} = Y_{WIN} + \Delta Y_{WIN}$;
- not adjacent to the window, if none of the previous conditions apply.

As CWs are found, they are pushed into the First-In-First-Out (FIFO) buffers and finally stored into the CW Serial-In/Parallel-Out (SIPO) register. FIFOs are necessary because in the parallel processing, more than one CWs can be recognized in the P_{AC} windows processed.

2) Near-window Check: The NWC extracts the Near Windows. A NW is defined as a window that could interfere with the expansion of the CW. In order to save time, we carry out the NW search together with the CW search, done by the AC block. As the CW is not identified yet, NW are recognized only as a function of the new DT. Figure 10 shows the worst-cases for the possible CW expansions, in terms of number of NWs. The orange areas indicate the pixels which must be free from NWs for a given expansion type. For the LD expansion represented in Figure 10d, the bottom side of the CW is not checked for NWs because those pixels are not yet explored: the DT is the last explored pixel in raster-order. The overall area to check for NWs corresponds to the union of the orange areas in the four expansion types, which is equivalent to the orange area for the RD expansion, united the orange area for the LD expansion. The maximum number of NWs for each DT is an important parameter to derive in order to optimize the processing. Figure 11 shows the worst-case configuration for the case $W_{MAX} = 8$. Basically, one non-expandable window plus three 1×1 windows can be fitted into the orange area, 4 NWs per side, therefore maximum 8 NWs for $W_{MAX} = L_{MAX} = 8$. The empty pixels between NW2,

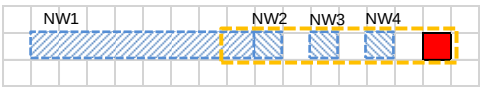

Fig. 11. Worst-case for the number of NWs

NW3 and NW4 are necessary for the worst-case, otherwise the NWs would not have been separate. The general formula that can be derived from the worst case is:

$$\mathrm{NW_{MAX}} = \left\lceil \frac{W_{MAX}}{2} \right\rceil + \left\lceil \frac{H_{MAX}}{2} \right\rceil \tag{5}$$

Similarly to the AC block, NWs are pushed into the FIFO buffers and finally stored into the NW SIPO register. FIFOs are necessary because in the parallel processing, more than one NWs can be recognized in the P_{AC} windows processed.

3) Expansion Check: Given a certain candidate window, the EC block ensures that the window is expandable without overlapping with other windows. If the candidate window is expandable, the result is passed to the Priority Selection block.

Given $(X_C, Y_C, \Delta X_C, \Delta Y_C)$ the coordinates of the candidate window and $(X_N, Y_N, \ \Delta X_N, \Delta Y_N)$ the coordinates of a near window, the following values are calculated in the EC block:

δ_L $= X_C - (X_N + \Delta X_N)$, the distance between the left border of the candidate window and the right border of the near window;

δ_R $= X_N - (X_C + \Delta X_C)$, the distance between the right border of the candidate window and the left border of the near window;

δ_U $= Y_C - (Y_N + \Delta Y_N)$, the distance between the upper border of the candidate window and the bottom border of the near window;

δ_D $= Y_N - (Y_C + \Delta Y_C)$, the distance between the bottom border of the candidate window and the upper border of the near window.

The distances δ_L, δ_R, δ_U and δ_D are depicted in Figure 12. For the sake of clarity, in Figure 12 the distances are represented for different near windows, with a directed arrow showing the direction of the positive value. During the execution of

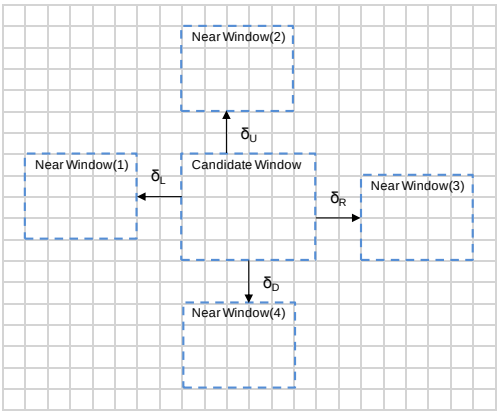

Fig. 12. Expansion check values

the algorithm, all δ_L, δ_R, δ_U and δ_D are evaluated for every window.

For every near window, the flags R, RD, D, LD are calculated, to evaluate which expansions are possible for the candidate, according to the following conditions:

$$R = (\delta_R \geq 1) \,||\, (\delta_L \geq 0) \,||\, (\delta_U \geq 0) \,||\, (\delta_D \geq 0) \quad (6)$$

$$RD = (\delta_R \geq 1) \,||\, (\delta_L \geq 0) \,||\, (\delta_U \geq 0) \,||\, (\delta_D \geq 1) \quad (7)$$

$$D = (\delta_R \geq 0) \,||\, (\delta_L \geq 0) \,||\, (\delta_U \geq 0) \,||\, (\delta_D \geq 1) \quad (8)$$

$$LD = (\delta_R \geq 0) \,||\, (\delta_L \geq 1) \,||\, (\delta_U \geq 0) \,||\, (\delta_D \geq 1) \quad (9)$$

where $||$ is the `or` operator. These conditions hold because overlapping is excluded. The flags are generated for every NW and the overall flags for the CW are calculated as the `and` of all the NW specific flags. For the overall flag, quadrant boundary conditions and maximum window size must be anded.

IV. Results

A. Timing model

Figure 13 shows the timing model of the CA in a block-diagram style. In order to set the parallelism degree for the architecture, each block is characterized by a worst-case execution time, for a clock frequency of 50 MHz, which is a reasonable target for RTAX2000 implementation. The timing block diagram describes the timing sequence of the different algorithm sections. Each section S is characterized by the worst-case number of cycles required for the execution, n_S. For the window load section:

$$n_{WL} = L_{WL} \cdot N_{DT} \quad (10)$$

where $L_{WL} = 1$ is the latency of the window load section and N_{DT} is the maximum number of DTs.

In the AC section, the AC block and the NWC block run in parallel in the same time. The timing estimation for this section can be calculated as:

$$n_{AC} = AC_i + L_{AC} \cdot N_{DT} \quad (11)$$

where $L_{AC} = 2$ is the latency of the AC section and AC_i is the number of iterations of the AC section. For each DT, the number of iterations AC_i of the AC/NWC section is roughly given by $AC_i \simeq \lceil N_{WIN}/P_{AC} \rceil$, where N_{WIN} is the maximum number of windows that the clustering algorithm can define and P_{AC} is the parallelism of the AC/NWC section. This formula assumes that N_{WIN} windows have to be checked since the beginning, that is a large approximation. A more accurate worst-case formula assumes that a new window is opened for each of the first N_{WIN} DTs:

$$\begin{aligned} AC_i &= (N_{DT} - N_{WIN}) \cdot \left\lceil \frac{N_{WIN}}{P_{AC}} \right\rceil + \sum_{i=1}^{N_{WIN}-1} \left\lceil \frac{i}{P_{AC}} \right\rceil = \\ &= (N_{DT} - N_{WIN}) \cdot \left\lceil \frac{N_{WIN}}{P_{AC}} \right\rceil + \\ &+ P_{AC} \cdot \left\lceil \frac{N_{WIN} - 1}{P_{AC}} \right\rceil \cdot \frac{\left\lceil \frac{N_{WIN}-1}{P_{AC}} \right\rceil + 1}{2} \end{aligned} \quad (12)$$

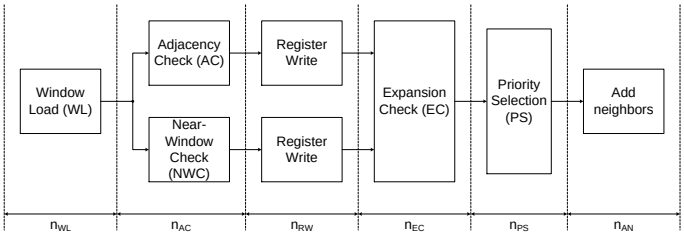

Fig. 13. Clustering Algorithm timing model

The register write section consists in writing the CW and NW values to the window registers for EC processing. This operation is usually done during the AC processing, but in the theoretical worst case that all CWs/NWs are concentrated at the end, $P_{AC} + 1$ cycles are required. Therefore, for the maximum theoretical worst-case, we can include these cycles for every DT:

$$n_{RW} = P_{AC} + 1 \quad (13)$$

The EC section compares the CWs with the NWs. Each unit processes one CW and a number of NWs dependent on the latency of the block, L_{EC}. The number of NWs processed in parallel is $N_{NW,EC} = \lceil NW_{MAX}/L_{EC} \rceil$. The execution cycles required by the EC block is given by:

$$n_{EC} = L_{EC} \cdot N_{DT} \quad (14)$$

For the Priority Selection block, given L_{PS} the latency of the unit, we have:

$$n_{PS} = L_{PS} \cdot N_{DT} \quad (15)$$

Finally, after all the DTs in a frame have been processed, the windows are sent to the next block, while adding the neighbouring one-pixel frame around the window.

$$n_{AN} = N_{WIN} + L_{AN} \quad (16)$$

Parameter	Description	Value (frame)	Value (quadrant)
N_{DT}	Maximum number of DTs	1000	813
N_{WIN}	Maximum number of windows	300	113
L_{WL}	Latency of window loading	1	1
L_{AC}	Latency of the AC block	2	2
P_{AC}	Parallelism of the AC and NWC blocks	8	4
L_{EC}	Latency of the EC block	2	8
L_{PS}	Latency of the Priority Selection (PS) block	2	2
NW_{MAX}	Maximum number of NWs	8	8
L_{AN}	Latency of the final window transmission and neighbour inclusion process	2	2

TABLE I
Parameters of the timing model

		L_{EC}			
		1	2	4	8
P_{AC}	2	101,1%	102,7%	105,9%	112,4%
	4	62,0%	63,6%	66,8%	73,4%

(a) Quadrant clustering

		L_{EC}			
		1	2	4	8
P_{AC}	2	273,9%	275,9%	279,9%	287,9%
	4	150,4%	152,4%	156,4%	164,4%
	8	95,7%	97,7%	101,7%	109,7%

(b) Full-frame clustering

TABLE II
RESULTS OF THE TIMING MODEL

The total worst-case execution cycles is given by

$$n_{TOT} = n_{WL} + n_{AC} + n_{RW} + n_{EC} + n_{PS} + n_{AN} \quad (17)$$

The number of available cycles per frame n_A, considering a processing clock frequency $f_{ck} = 50\,\text{MHz}$ and a frame time $T_f = 1\,\text{ms}$ is:

$$n_A = \lfloor T_f \cdot f_{ck} \rfloor = 50000 \quad (18)$$

The parameters of the timing model are summarized in Table I, for both the full-frame clustering and quadrant-specific clustering.

The architecture is parametric with two separate degrees of freedom:

- P_{AC} is the number of windows processed in parallel in the AC/NWC sections.
- L_{EC} is the latency of the EC block, which defines the number of NWs processed in parallel by the EC.

The other latencies are parametric as well, but are fixed due to timing closure, to reach the target clock frequency on the RTAX FPGA. The results of the timing model for the parameters P_{AC} and L_{EC} are represented in Table II, for the quadrant and frame versions of the algorithm, respectively. The algorithm execution time is expressed as a percentage of the single frame time, fixed to 1 ms. The algorithm execution time needs to be lower than the frame time (¡100%) for the system to meet the requirements and to be considered real time.

B. Synthesis Results

In this section, the synthesis results on the Microsemi RTAX2000S FPGA [10] for both quadrant clustering and full-frame clustering are reported. The RTAX2000S have 21504 combinational resources, 10752 registers and 288 Kbits of RAM available to the user. The synthesis results are given as a function of the architecture parallelism at AC level, P_{AC}, and the parallelism at EC level, L_{EC}. The HW resource occupation increases with the increasing of P_{AC}, and decreases with the increasing of L_{EC}, as the latter parameter represents the latency of the EC block, i.e., the lower L_{EC}, the higher the parallelism.

		L_{EC}			
		1	2	4	8
P_{AC}	2	75,1%	64,2%	55,2%	49,4%
	4	89,6%	79,1%	69,6%	63,9%

(a) Combinational

		L_{EC}			
		1	2	4	8
P_{AC}	2	28,6%	29,1%	28,9%	28,7%
	4	32,9%	33,3%	33,1%	33,0%

(b) Registers

		L_{EC}			
		1	2	4	8
P_{AC}	2	37,5%	37,5%	37,5%	37,5%
	4	75,0%	75,0%	75,0%	75,0%

(c) RAM

TABLE III
RTAX2000S SYNTHESIS RESULTS FOR QUADRANT CLUSTERING

Table III shows the synthesis results for the quadrant clustering, as a function of P_{AC} and L_{EC} parameters. The timing analysis in Table II shows that all the solutions with $P_{AC} = 4$ are good. Therefore, the solution in the case of quadrant clustering is given by $P_{AC} = 4$, $L_{EC} = 8$, minimizing HW resource occupation.

This configuration gives an occupation of 63.9 % combinational cells, 33.0 % registers, 75.0 % RAM, and has a worst-case execution time of 73.4 % frame-time.

Table IV shows the synthesis results for the full-frame clustering. From the timing analysis shown in Table II, only the solutions with $P_{AC} = 8$ and $L_{EC} = \{1,\ 2\}$ are good. Therefore, the solution in the case of full-frame clustering is given by $P_{AC} = 8$, $L_{EC} = 2$, minimizing HW resource occupation.

This configuration gives an occupation of 29.4 % combinational cells, 11.5 % registers, 51.6 % RAM, and has a worst-case execution time of 97.7 % frame-time.

Compared to the quadrant clustering, the full-frame clustering is much more optimized in terms of HW resource occupation. This result was predictable, since the worst-case concentration of DTs forces to have the quadrant clustering nearly as powerful as the whole frame clustering. In fact, the single quadrant clustering must be able to process 813 DTs in the frame time, with a maximum number of windows of 113, while the full-frame version has to deal with 1000 DTs and 300 windows.

The final architecture parameters are shown in Table I.

V. Conclusion

In this paper the problem of data clustering for the Lighting Imager has been examined. First of all a preliminary analysis has been carried out to understand the possible saving in terms of data-rate with the insertion of a clustering algorithm,

		L_{EC}			
		1	2	4	8
P_{AC}	2	20,8%	17,8%	15,1%	13,6%
	4	24,8%	21,8%	19,2%	17,6%
	8	32,5%	29,4%	26,8%	25,2%

(a) Combinational

		L_{EC}			
		1	2	4	8
P_{AC}	2	8,0%	8,1%	8,0%	8,0%
	4	9,2%	9,3%	9,2%	9,2%
	8	11,4%	11,5%	11,5%	11,4%

(b) Registers

		L_{EC}			
		1	2	4	8
P_{AC}	2	17,2%	17,2%	17,2%	17,2%
	4	26,6%	26,6%	26,6%	26,6%
	8	51,6%	51,6%	51,6%	51,6%

(c) RAM

TABLE IV
RTAX2000S SYNTHESIS RESULTS FOR FULL-FRAME CLUSTERING

together with a literature analysis on the different clustering possibilities. A novel algorithm has been proposed, both in a quadrant specific and in a full frame version, in comparison with the algorithm available in literature. The algorithm has been mapped creating a block level system architecture. Keeping in mind the system timing constrains, fixed at 50MHz on the Microsemi RTAX2000 FPGA, a timing model has been proposed. The target of this analysis was to show which parameters of the two identified algorithm guarantee that the execution time stays under the frame time, as required at system level. Finally a synthesis has been performed on these architectures. From the results of this study, it is possible to notice that, once the timing constraints is met, full-frame clustering is much more optimized in terms of HW resource occupation. In future work the algorithm could be improved further employing more than one FPGA, exploiting advantages coming from parallelisation. Also a more sophisticated testing environment could be created, taking advantages of the ESA and NASA Earth Observation catalogue available online.

ACKNOWLEDGMENT

This research project was funded by Leonardo S.p.A., Campi Bisenzio, Firenze, Italy.

REFERENCES

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, Sep 1999. [Online]. Available: http://dx.doi.org/10.1145/331499.331504

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, E. Simoudis, J. Han, and U. M. Fayyad, Eds. AAAI Press, 1996, pp. 226–231.

[3] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 2007*, 2007. [Online]. Available: http://dx.doi.org/10.1145/1281192.1281210

[4] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: theory and practice," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 3, pp. 515–528, May 2003. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2003.1198387

[5] C. C. Aggarwal, T. J. Watson, R. Ctr, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th VLDB Conference*, 2003, pp. 81–92.

[6] A. Magdy, N. A. Yousri, and N. M. El-Makky, "Discovering clusters with arbitrary shapes and densities in data streams," *2011 10th International Conference on Machine Learning and Applications and Workshops*. [Online]. Available: http://dx.doi.org/10.1109/ICMLA.2011.56

[7] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *In 2006 SIAM Conference on Data Mining*, 2006, pp. 328–339.

[8] P. Hore, L. Hall, D. Goldgof, and W. Cheng, "Online fuzzy c means," *NAFIPS 2008 - 2008 Annual Meeting of the North American Fuzzy Information Processing Society*. [Online]. Available: http://dx.doi.org/10.1109/NAFIPS.2008.4531233

[9] L. Tommasi, G. Basile, A. Romoli, and M. Stagi, "Design and performance of the lightning imager for the meteosat third generation," in *Proc. 6th Internat. Conf. on Space Optics*, 2006.

[10] "Rtax-s/sl and rtax-dsp radiation-tolerant fpgas datasheet." https://www.microsemi.com/product-directory/rad-tolerant-fpgas/1694-rtax-s-sl, accessed: 2019-04-29.

[11] S. Lorenzini, R. Bardazzi, M. D. Giampietro, M. Taccola, L. P. Cuevas, and F. Feresin, "Optical design of the lightning imager for mtg," in *International Conference on Space Optics 2012*, 2012.

[12] S. Parkes and P. Armbruster, "Spacewire: A spacecraft onboard network for real-time communications," vol. 2005, 2005, pp. 6–10, cited By 36. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-33751412258&doi=10.1109%2fRTC.2005.1547397&partnerID=40&md5=2d66d04e4428eb62d6e270a10d55a61e

Pietro nannipieri Pietro Nannipieri graduated cum Laude in Electronic Engineering (MSc) at the University of Pisa in June 2016. During his bachelor degree he has been a visiting student at University College of London for one year. His interests are digital and VLSI design as well as electronics for space applications. He took part in the 18th and 21st ESA Rexus Program with PHOS team as electronic engineer and with U-PHOS Team as project manager. He is a PhD student in the VLSI lab of the information engineering department, University of Pisa and he his currently a visiting researcher at ESTEC (ESA). His work mainly focus on the development of IPs for satellite on-board data handling, (i.e. SpaceFibre), but also on signal processing.

Daniele Davalle has got the M.Sc. degree in Electronic Engineering from the University of Pisa in 2011 (110 cum laude). He received the Ph.D. degree in 2015, with a thesis on electronic systems design for on-board satellite digital signal processing and networking, at the Department of information Engineering, University of Pisa. His research interests are mainly focused on algorithm/architecture co-design for communication, image processing and networking systems. His expertise ranges from system specification definition, algorithm modelling and simulation to the complete hardware implementation flow for Xilinx and Altera FPGAs and up to the logic synthesis for Microsemi RTAX FPGAs and for ASICs. He is advisor of some Master thesis in Electronic Engineering in the field of VLSI architectures for digital signal processing and networking. He is member of the SpaceWire and SpaceFibre working groups and he is involved in the SpaceFibre standardization committee. In 2014 he has been visiting researcher at the European Space Agency. He has worked also as contractor for Consorzio Pisa Ricerche.

Stefano Nencioni Stefano Nencioni got the Laurea in Electronic Engineering from University of Florence in 1998. From 1999 to 2001, he was with the SIRIO PANEL, as Electronic Designer of part of the Cockpit for Aircraft and Car. He is Electronic Digital Designer at Leonardo Company. He is involved in several Projects, ViR for DAWN, ALADIN, JIRAM for JUNO, as FPGA designer and Electronic Engineer Leader. Currently he is involved as Electronic Engineer Leader in the Lightning Imager instrument.

Paolo Lombardi Paolo Lombardi graduated cum laude in Electronic Engineering at the University of Florence in April 1998. Since September 1998 is ASIC/FPGA specialist in the Airborne and Space Systems division of Leonardo SpA, with specific expertise and experience in applications for Space products.

Luca Fanucci Luca Fanucci got the Laurea and the Ph.D. degrees in Electronic Engineering from University of Pisa in 1992 and 1996, respectively. From 1992 to 1996, he was with the European Space Agency - ESTEC, Noordwijk (NL), as research fellow. From 1996 to 2004 he was a senior researcher of the Italian National Research Council in Pisa. He is Professor of Microelectronics at the University of Pisa. His research interests include several aspects of design technologies for integrated circuits and electronic systems, with particular emphasis on system-level design, hardware/software co-design and sensor conditioning and data fusion. The main applications areas are in the field of wireless communications, low power multimedia, automotive, healthcare, ambient assisted living and technical aids for independent living. He is co-author of more than 400 journal and conference papers and co-inventor of more than 40 patents. He served in several technical programme committees of international conferences. He was Program Chair of DSD 2008 and DATE 2014 and General Chair of DATE 2016. He is a member of the editorial board of IOS Press Technology and Disability Journal. He is a Senior Member of the Institute of Electrical and Electronic Engineers.