Edge computing for space applications: Field programmable gate array-based implementation of multiscale probability distribution functions

Norbert Deak,[1,a)] Octavian Creţ,[1,a)] Marius Echim,[2,3,a)] Eliza Teodorescu,[2,a)] Cătălin Negrea,[2,a)] Lucia Văcariu,[1,a)] Costel Munteanu,[2,a)] and Anca Hângan[1,a)]
[1]*Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania*
[2]*Institute of Space Science, Măgurele, Romania*
[3]*Royal Belgian Institute for Space Aeronomy, Brussels, Belgium*

(Received 12 June 2018; accepted 22 November 2018; published online 12 December 2018)

Data processing is a challenging problem in space applications. The limited bandwidth available for communication between satellites and the ground and the increasing resolution of scientific instruments make it virtually impossible to transfer all the data recorded on board. Although various mitigation strategies were developed, large amounts of on-board data are still lost. This paper presents a Field Programmable Gate Array (FPGA)-based architecture which is able to perform on-board nonlinear analysis of data and compute probability distribution functions of fluctuations. We propose two implementations for our solution, which can be used for space applications and also other computational contexts. On a spacecraft, the logic resources of the FPGA will typically be shared by several designs running various digital signal processing algorithms. That is why each algorithm should be designed in variants, optimized for different criteria, so that the entire group of algorithms makes an efficient usage of the FPGA resources. The proposed solution focuses on two major optimization criteria, area and speed, such that the FPGA resources are efficiently used. Also, the power consumption is at least two orders of magnitude less in comparison with classical software implementations. The solution was tested with both synthetic and real data and shows excellent results paving the way towards an application that can be ported on a space-grade FPGA. *Published by AIP Publishing.*
https://doi.org/10.1063/1.5044425

I. INTRODUCTION

Increasing performance, both in terms of quantization and time resolution, of state-of-the-art scientific instruments raises several challenges related to data processing. The problem is even more complex in space exploration, where, due to limited telemetry bandwidth, it is impossible to transfer to the ground the entire amount of data recorded on board. Various mitigation strategies were developed; however, they solve the problem only partially. In other words, although the scientific instrument is able to collect a large amount of samples, only a fraction of these data is transmitted to the ground. Here we discuss a hardware solution which uses the advantages of Field Programmable Gate Arrays (FPGAs) for the on-board analysis of data.

The FPGA-based implementation discussed here is tailored for the nonlinear analysis of plasma data and provides estimates of a probabilistic measure of variability: the probability distribution function (PDF) of events. A probabilistic description of variability based on PDFs provides a detailed description of the multi-scale structure and topology of fluctuations. However, such a level of detail comes at quite high cost; one needs a sufficiently large and representative statistical ensemble of samples. Nevertheless, in general, only a limited number of PDFs moments can be computed.[1]

Our solution can be considered as a system design strategy that is subsumed by the concept of edge computing: moving the computation part near the data source instead of sending the data for processing to the central unit on the ground.[2]

Recently, the idea of on-board data analysis was adopted by NASA's Magnetospheric Multiscale Mission (MMS). This multi-spacecraft mission collects high resolution (burst mode) data for half of the time of an orbit; these data are stored in the on-board memory (flash-based RAM). However, only a small fraction, approximately 2%, is subsequently transmitted to the ground due to bandwidth limitations.[3] The decision on which data to be transferred to the ground is taken semi-automatically based on some on-board data analysis and the input from an operator from the ground (the scientist in the loop—SITL[3]). The data quality values are determined from complex multi-criteria triggers provided by the on-board instruments. However, this automatic procedure can be overwritten by the SITL who inspects visually the on-board data and other data sources to decide on the scientifically interesting time intervals.

Both strategies, however, do not increase the overall percentage of data available to scientists on the ground. The development of the FPGA-based system discussed in this paper provides a solution to exploit the entire amount of data stored on the spacecraft's flash-based RAM. We built an FPGA-based hardware solution which is able to compute on-board the PDFs

a)Norbert.Deak@cs.utcluj.ro; Octavian.Cret@cs.utcluj.ro; echim@spacescience.ro; eliteo@spacescience.ro; negreacatalin@spacescience.ro; Lucia.Vacariu@cs.utcluj.ro; costelm@spacescience.ro; and Anca.Hangan@cs.utcluj.ro

of a time series (be it magnetic field, plasma moments, or electric field). The benefit of this approach is three fold:

(1) Some key aspects of the analysis of PDFs computed on-board can be integrated in the loop as additional "event" descriptors/triggers to be considered by SITL in MMS-like strategies for data selection.
(2) Large fractions of data (otherwise lost since they are not sent to the ground) are analyzed on-board and the variability depicted by these data is quantified; this result extends considerably the scientific impact of the mission.
(3) The PDFs computed on-board represent in fact a compressed image of data variability. Indeed, a data stream comprising, say, 10^7 samples leads to a number of K PDFs revealing the variability at K different scales; each PDF comprises a maximum number of 100 values. In general, K is less than 10. Thus, the variability of the 10^7 samples is quantified by 10×100 numbers that represent an effective data compression at a rate of 10^4.

The main contributions highlighted in this paper are:

- The development of a PDF computation architecture implemented on an FPGA device, focusing on minimizing the utilization of logic resources [slices, block random accessed memory (BRAMs), flip-flops, etc.] available in Xilinx FPGAs.
- The development of a data acquisition block, part of the above-mentioned architecture, which is capable of storing all the necessary data using the least amount of FPGA logic slices.
- Two implementation variants of the histogram computation block, focused on minimizing the resource usage and maximizing the maximum operating frequency, respectively.
- A solution that reduces the power consumption compared to software [personal computer (PC) or microcontroller-based] implementations of the same algorithm.
- The implementation of a communication protocol with a PC/magnetometer for testing, with synthetic and real magnetometer data.

The paper is organized as follows. Section II gives a brief theoretical background and a survey of relevant previous studies. The implementation of the proposed architectural solution is presented in Sec. III. Section IV discusses the experimental tests and results, while Sec. V concludes this work.

II. THEORETICAL BACKGROUND AND PREVIOUS WORK

The probability density function f offers a natural description of the distribution of a random variable, X, and gives the probability for X to take values within the limits a and b.

The histogram method is probably the most popular algorithm to compute the probability density function of a variable X from a time series of N samples. It splits the input data domain into M bins[4] and uses a binning defined by $[x_0 + mh, x_0 + (m+1)h]$, where x_0 is the origin and h is the bin width; m takes values from 1 to M. The value of the PDF associated with the bin Δx_m is defined as[4]

$$PDF(\Delta x_m) = \frac{1}{Nh}(no\ of\ X\ in\ bin\ \Delta x_m). \tag{1}$$

Given a time series of the variable X, a measure of the variability at a time t and the temporal/spatial scale τ is given by the difference

$$\Delta X(t)|_\tau = X(t+\tau) - X(t). \tag{2}$$

When the observations are provided at a time resolution δ, one can write $\tau = j\delta$, with j being an integer number. In general, the scales increase in bytes such that $j = 2^k$, with k being a natural number. Since the time series is discrete, Eq. (2) can be rewritten as

$$\Delta X(i)|_\tau = X(i+j) - X(i). \tag{3}$$

For each scale τ (or scale index $j = 2^k$), one creates a statistical ensemble of fluctuations, $\{\Delta X_i\}|_\tau$, from values of $\Delta X(t)|_\tau$ evaluated for each sample in the time series. The PDFs are computed by applying the histogram method on the statistical ensemble of fluctuations at each scale. If X is a vectorial variable (e.g., magnetic field, plasma bulk velocity), the PDFs are computed for each component of the field. Note also that there is a maximum scale $\tau_{MAX} = 2^{k_{MAX}}\delta$ for which one can compute the PDF from a time series of total length N. The value of τ_{MAX} is determined such that the statistical ensemble $\{\Delta X_i\}|_{\tau_{MAX}}$ still includes a large enough number of values. In the reminder, we consider a time resolution equal to $\delta = 1$ s, and thus the scale (in seconds) is numerically equal to $\tau = j = 2^k$.

The probabilistic description of data variability based on PDF analysis is needed to investigate fundamental processes, for instance, space plasma turbulence. Until now, such analyses were performed exclusively on the ground, on calibrated data. Probabilistic results on variability were published for solar wind (e.g., Refs. 5 and 6) as well as terrestrial[7] and planetary[8] plasma data. The starting point of our efforts is represented by a complex data analysis tool, the Integrated Nonlinear Analysis (INA) library,[9] which cumulates several nonlinear data analysis tools and algorithms, from lower to higher order methods. INA adopts a simple algorithm for assessing PDFs based on the histogram of fluctuations at various scales. The FPGA-based solution presented here adopts the same methodology, which is discussed in Sec. III.

Edge computing is a novel paradigm for pushing the computation near the data source (usually sensors), as presented in Refs. 2, 10, and 11. It can answer the problems raised, for instance, by the cloud computing paradigm.

Moving the data processing towards the edge, or sensor devices, requires more processing capabilities, but at a lower energy consumption. FPGAs seem to be the perfect candidates.[12] However, this can transform the design process into a challenging task because it implies both high-level software and low-level hardware design. In Ref. 12, the author proposes a homogeneous design methodology and environment to solve this challenge using object-oriented design (OOD) and addresses all parts of the system: for hardware design, the object oriented principles and patterns are used to improve reusability, adaptability, and extensibility; for hardware/software co-design, OOD is used to abstract their

integration and communication; the middleware realizes the integration of the device in the network and also allows remote reconfiguration.

Even without the edge computing paradigm, FPGAs were considered for sensor nodes, either as standalone platforms or with a microcontroller, as presented in Ref. 13. The low power consumption and the computation speed offer a great enhancement in comparison to microcontrollers of commercial sensor nodes.

Due to the high computational performance, the reconfigurability, and the low energy consumption of the FPGAs, the author in Ref. 14 investigates applications of FPGAs for space systems. Thorough analysis of radiation effects for existing devices are presented, proving that a multi-redundant "2-out-of-*n*" system based on static random access memory (SRAM)-FPGAs and a flash-FPGA voter has the highest reliability against radiation effects for low earth orbit missions.

Numerous studies addressed the problem of histograms or PDF designs for FPGAs due to their applicability in various scientific domains. In Ref. 15, the author presents an estimator architecture designed to dynamically create a real time histogram of the values of a time series. A single histogram is estimated on a step-by-step basis and the results are available in real time. The design is also aimed to optimize the area (the amount of logic resources in the FPGA chip). Therefore, the histogram is stored in counters (implemented using slice flip-flops), while a Block Random Accessed Memory (BRAM) memory block contains the enable signals to these counters. There are 2^l bins for the histogram, so to address the histogram counters, simply the l most significant bits of the input samples are used. To parallelize this process, dual-ported BRAMs are used to read two addresses at the same time. When a new data sample is received, the histogram bin is updated, the cumulative distribution function (CDF) is computed, and updated statistical information becomes available.

Another work, although devoted to digital image processing, uses BRAMs to store the histograms.[16] The goal is to obtain an FPGA-based implementation of parallel histogram equalization for images. In this work, a single histogram is computed for the two-dimensional image. For each pixel, the histogram bin is read, incremented, and written back in the BRAM. To speed up the implementation, each image pixel is evaluated in a single clock cycle. The authors make this possible by using dual-ported BRAMs: the current value of the histogram bin is read through the first port, then it is incremented, and finally it is written back through the second BRAM port. For a greater speed-up, multiple BRAMs are used to process more input pixels in parallel.

Reference 17 presents a method which implies storing the histogram bins in BRAMs and evaluating one data sample in each clock cycle. The authors use two clock domains: one for the input samples and a faster one to update the histogram counters. The process of incrementing the values in each bin is divided into two sub-cycles: a *read cycle* for getting the prior value and a *write cycle* to write back the incremented one.

The idea from Ref. 16 is also discussed in Ref. 18 and in a Xilinx White Paper,[19] which details different usages of BRAMs. It presents the exact same approach of updating a value, i.e., performing a *read-modify-write* operation in one clock cycle. It specifies that port A of the BRAM block must be used as a read port, while port B must be used as a write port. A common clock must be used, and the write address, together with the modified data from port A, is delayed with one clock cycle before arriving to the write port.

These previous FPGA designs are not appropriate for the complexity of our scientific objective. Indeed previous solutions computed one single histogram for the *values* of the variable, and here we compute several probability distribution functions, each derived from a normalized histogram of the *fluctuations* of the variable for different scales. The big challenge from the point of view of hardware development with FPGAs is to collect/store all the samples on the FPGA chip until a sufficient number is available to compute the differences described by Eqs. (2) and (3). The latter are indeed a measure of fluctuations for the different scales τ. The histograms are then computed for the distribution of the amplitudes of these differences (*deltas*). For a large τ, this task can be difficult; e.g., for a scale $2^{k_{MAX}} = 8192$, at least 8192 points must be kept on the FPGA at the same time. In Refs. 15 and 17, input data are stored in BRAMs—a solution that is not suitable for our case, as explained in Sec. IV A. In Ref. 16, input data are not stored on the device (only the histogram computation part resides on the FPGA chip). Moreover, as for the histogram computation part, none of these designs consider the two issues described by us below (in Sec. IV B), which leads to inaccurate results if ignored. So these designs cannot be adapted to compute a histogram-based PDF on the data differences estimated at several different scales.

III. IMPLEMENTATION

The implementation of an FPGA-based application depends strongly on the context. An FPGA chip placed on a satellite would be requested to perform several types of analyses of the targeted data samples, based on various digital signal processing (DSP) algorithms, like Fast Fourier Transform (FFT), PDFs, and their moments (flatness), wavelet, etc. This requires that most of the optimization effort will be oriented towards minimizing the occupied area on the FPGA chip. Here we work with the scenario that the data to be analyzed on-board are vectorial (e.g., provided by a three-axis magnetometer). We consider that a total number of $N = 10\,000$ sample points are available to compute the PDF for each component; these data need to be stored on the FPGA chip. The differences (3) are computed for each sample and for each scale $\tau = \{1, 2, \ldots, 8192\}$ to create the corresponding statistical ensemble of differences for each scale. According to Eq. (3), these differences are: $B(0) - B(1)$, $B(0) - B(2)$, $B(0) - B(4)$, $B(0) - B(8)$, $\ldots$, $B(0) - B(8192)$. Then the PDFs are computed with the histogram method applied on the ensemble of these differences.

The magnetometer's data acquisition rate (typically between 66 and 100 Hz for fluxgate sensors) is at least three orders of magnitude less than the regular operational frequency of the FPGA device (hundreds of MHz). This very important aspect should be taken into consideration while designing the

solution. Consequently, the system can perform many computations between the arrivals of two consecutive data samples gathered by the magnetometer, so the main optimization criterion becomes the occupied area (speed is not a real problem in this context).

The algorithm to be implemented is described below (Algorithm 1). In the rest of the paper, we assume that the data (B) are provided by a magnetometer; however, any other type of variable, gathered by other types of sensors, can be considered.

Most FPGA chips are intended for classical applications (rapid prototyping in various fields, such as education, etc.), but there is a special category of chips that are specially designed for space applications—the so-called "space-grade" (Xilinx) or "rad-tolerant" (Microsemi) FPGAs. Such an FPGA chip is manufactured in a technology that can still work correctly in the extreme conditions encountered in space (high level of radiations, extreme temperatures, etc.). The manufacturing technology is completely transparent to the user, who is not aware of the technology the FPGA chip is built in, and who designs the application at the logic level. For instance, the Xilinx manufacturing process implies (among other techniques) applying a thin epitaxial film in the wafers in order to prevent latch-up effects, while Microsemi mainly uses antifuse technology.

A major difference between space-grade and classic FPGA chips is the fact that in the former chip, all logic elements are triplicated and a voting mechanism is hardwired inside it to ensure that in case of a disturbing event (for instance, SEU—*single event upset*), the design's functioning is not affected. This technique is called Triple Modular Redundancy (TMR). TMR can be hardwired in silicon (like in Microsemi's RTAX-S radiation-tolerant FPGAs), or the FPGA manufacturers can provide software tools that support automatic design triplication (like the TMRTool from Xilinx).

Besides that, the logic structure of its internal resources is mostly the same as the one of normal FPGA chips. Therefore we tested our solution on laboratory FPGA technology as a first step towards prototyping a space-qualified solution. The main novelty at this step is the conceptual and algorithmic design of computations typical for the probabilistic description of data variability; later on, the solution can be scaled to actual resources available from space-qualified chips.

For our design, we have used a Xilinx FPGA chip from the Artix7 family. A Xilinx FPGA chip usually contains four major types of hardware resources: *logic slices*—which are used for implementing either Boolean functions, memory blocks (called "distributed memory"), or small first in, first out (FIFO) memories/shift registers; *slice flip-flops*—each slice contains up to 8 flip-flops that can be used for implementing data registers; *BRAMs*—18/36 kbit memory blocks with dual port capabilities; and *DSP blocks*—which can implement basic multiply and accumulate operations often used in DSP applications.

ALGORITHM 1. Histogram computation for K different scales.

1. **For each** t **from** 0 **to** N
2. *Read* measured data sample $\boldsymbol{B}(t)$ at data acquisition rate (CLK_A)
3. *Save* it in the corresponding resource
4. **For each** τ between 1 and K
5. *Compute* $\boldsymbol{B}(t) - \boldsymbol{B}(t-\tau)$
6. *Increment* the corresponding bin
7. **end for**
8. **end for**

Since the architecture used to perform the PDF analysis will "share the space" with other architectures designed to perform other types of analysis (e.g., FFT, flatness, etc.), we must consider solutions that minimize the occupied area. Since each of these architectures usually makes use of a specific type of resources (some may use mainly slices, other may use mainly BRAMs or DSP blocks, etc.), it is important to have several alternative solutions.

The proposed architectural solution implements the steps 2, 3, 5, and 6 of Algorithm 1: *data acquisition* and *histogram computation*.

As for the *data acquisition* part of Algorithm 1 (steps 2 and 3), the basic idea behind our architectural solution is to store all the data samples into look-up tables (LUTs) of each slice configured as shift registers using the SRL16 and SRLC32 macros and compute the differences at τ (*tau*) distances (step 5).

For the *histogram computation* part, we also propose two implementation variants, optimized for the two main design optimization criteria: the first one has a parallel architecture that increases the maximal operational frequency of the design, while the second one minimizes the occupied area in the FPGA chip. The histogram bins are stored in BRAM blocks. The basic ideas behind our architectural solutions are:

1. When a difference computed with (3) falls into a specific bin, the system reads the current value of the bin from the BRAM, increments it, and writes it back in the BRAM block in one clock cycle (more precisely, *using the same clock as the input data* and thus delaying the writing). In this case, the advantage is that there is a unique clock signal in the whole design, thus making it possible to achieve a higher operational frequency.
2. When a difference computed with (3) falls into a specific bin, the program reads the current value of that bin from a BRAM, increments it, and writes it back in the BRAM block with a *faster clock rate* than the data acquisition rate. The advantage is the smaller amount of resources that are necessary for the implementation (smaller area).

The block diagram of the whole system is shown in Fig. 1 and will be detailed in Subsections III A and III B.

A. Data acquisition block

In this block, all the data samples are stored on the FPGA chip in a long shift register, whose elements are actually FIFO LUTs from the slices configured as shift registers (SRL16 and SRLC32). This is a very effective way of using LUTs (one LUT is equivalent to up to 32 slice flip-flops). Since we only need the data samples at τ distance from the currently gathered data sample, it is enough to be able to access them only at the corresponding indices, as shown in Fig. 2 (the upper blocks are these FIFOs). The first FIFO's length is less than 32 ($x_0, x_1, x_2, x_3 - x_4, x_5 - x_8, x_9 - x_{16}, x_{17} - x_{32}$), so for them,

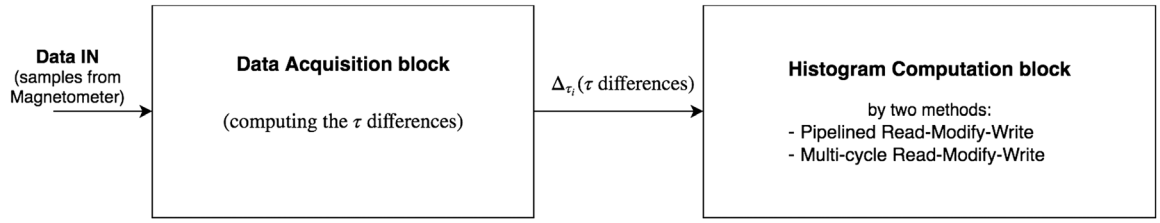


FIG. 1. The block diagram of the whole system.

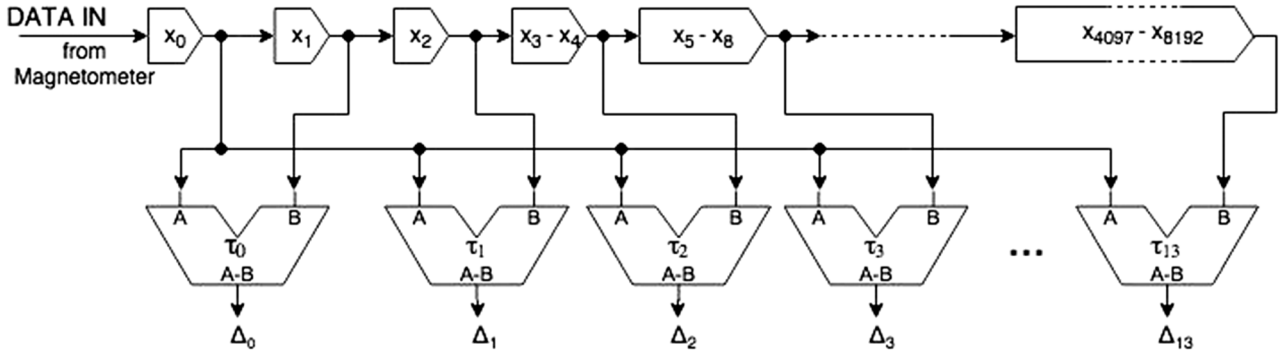


FIG. 2. FIFOs (LUTs configured as shift registers, SRL16/SRLC32) providing access at the data samples (x_0) with a step of τ.

SRL16 was used, but then for all the others, SRLC32 was instantiated.

In general, the input data samples are stored in BRAMs. Since our goal is to optimize the FPGA resources to be shared with other designs, we found a solution that that does not instantiate BRAMs. It has the advantage to render BRAMs available for the implementation of additional algorithms on the same FPGA chip.

An alternative to using BRAMs is to store the input data samples in slice flip-flops—a solution that does not fulfill the optimization criterion. The solution we adopted is based on shift registers built from FIFO LUTs; it reduces the occupied logic area by a factor that increases with the number of data samples and the number of τ scales implemented in the design.

If the data samples are stored in flip-flops, each data bit will be stored in a separate flip-flop. It is mandatory to store τ_{MAX} (the largest scale) data samples at the same time, and since data are 16 bits wide, the total amount of flip-flops needed is $\tau_{MAX} \times 16$. The FPGA device has eight flip-flops per slice, so the total amount of slices is $\tau_{MAX} \times 2$.

On the other hand, by instantiating the SRL16 and SRLC32 macros, only a much smaller number of slices are necessary. Each slice contains four LUTs, and in almost half of the slices on the FPGA, these LUTs can be configured as one-bit SRL16 or SRLC32 primitive shift registers. Table I compares the number of slices needed for both solutions.

In order to compute the PDFs for K different scales, it is necessary to simultaneously access $K = k_{MAX} + 1$ values. In our case $K = 14$. Note also that a solution based on BRAMs would allow only two simultaneous accesses to the stored value; therefore, the architecture would have been significantly more complex and would have consumed more resources. Thus, another advantage of our method is the fact that all K values are simultaneously accessible at any moment.

In the diagram illustrated in Fig. 2, x_0 represents the current sample received from the on-board scientific instrument, x_1 is the sample received at the previous time step, etc. (x_t is the sample received at time t). Each Δ stores the difference computed with Eq. (3) for the 14 different time scales 2^k with $k = 0 \ldots 13$. Δ is updated for each data reading and is computed in parallel for all the scales. The histogram for each scale $\tau = 2^k$ is also updated in parallel by increasing the counter of the bin where Δ falls.

TABLE I. Comparison of the required number of slices for the implemented method and of the alternative method for the data acquisition block.

	Flip-flop-based solution		LUTs as shift-registers (this solution)		Reduction
τ_{MAX}	# Flip-flops	# Slices	# LUTs (as SRL16 or SRLC32)	# Slices	factor
1024	16 384	2 048	608	152	13.47
2048	32 768	4 096	1 120	280	14.62
4092	65 536	8 192	2 144	536	15.28
8196	131 072	16 384	4 192	1048	15.63
65536	1 048 576	131 072	32 864	8216	15.95

As in our solution the occupied FPGA area consists of logic slices, the amount of resources of these types that remain available for implementing other designs is reduced.

This idea of using the LUTs as special shift registers is specific for Xilinx FPGAs. Also, if other designs need to use the stored data samples, they will be accessible only at the specified access points, at specific distances from 0 (these distances are given by the different values of τ).

B. Histogram computation block

As the main goal of this design is to compute $K = k_{MAX} + 1 = 14$ different histograms in parallel, the result can be compressed to obtain more meaningful information by varying the number of bins and their width. By working on 16 bit input data samples, the differences are also at most 16 bits wide, including positive and negative numbers (the differences are represented in two's complement numbering system). If we choose to have for instance $128 = 2^7$ bins, it is enough to crop out the 7 most significant bits of the Δ results, which will point to the corresponding bin.

The final result consists in a collection of M different bins, each of them containing the number of differences Δ falling in that given bin interval. The best way to store these numbers is in BRAMs, similar to Refs. 16 and 17. Each time a new Δ is computed, the algorithm finds in which bin it falls and that bin is incremented using a simple scheme like the one presented in the block diagram in Fig. 3.

A histogram is computed for each scale (there are K scales) and thus the block diagram shown in Fig. 3 is replicated $K = 14$ times in this design. Each histogram bin has a maximal capacity of $256 \times 16 = 4096$ bits; in case the chosen number of bins is 256 in total (128 for negative and 128 for positive values), each histogram (PDF) is stored in a separate BRAM block. Since a BRAM block may store up to 36 kbits, we can be sure that the histogram bin will fit in one BRAM, so in total K BRAMs are needed.

Updating the histogram bins may look like a simple task, but it presents some hidden difficulties. In order to increment a value from the memory, one first needs to read it out, increment it, and then write it back in the memory. These operations need to be performed for each newly gathered data sample—that is, in one clock cycle. But reading and writing from and to the BRAM takes at least one clock cycle separately, so some additional logic needs to be added to solve this problem. We propose two different architectural solutions: the first one features a pipelined read-modify-write operation *per* clock cycle and the second one features a read-modify-write operation using a supplemental, faster clock.

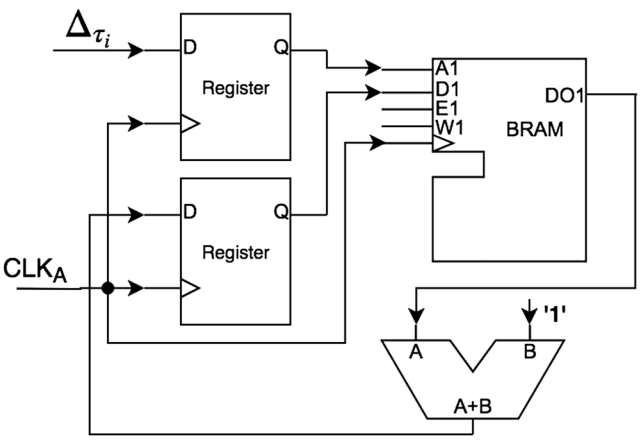


FIG. 3. Block diagram of the histogram computation for one τ.

1. Method 1 (pipelined read-modify-write memory operation)

The first solution consists of a pipelined and fully synchronized design (a unique clock signal, CLK_A, is used in the whole design). For each τ, a dual-ported BRAM is used to store the histogram bins. Each BRAM address stores a value that corresponds to a bin.

After Δ is computed, the result is stored in a data register. In the next clock cycle, the 7 most significant bits of the result constitute the address on the first BRAM port, so the corresponding value is extracted, incremented, and prepared to be written back into the BRAM. This address is propagated through a series of registers to have the correct address at the moment of writing on the second port (since there is a new address arriving in each clock cycle). Finally, the data are written back to the BRAM.

The design is broadly similar to the ones presented in Refs. 16 and 19; however, due to the pipelined structure, there are two cases when this design could fail. To prevent this, additional logic was added to cover each corner case. The detailed block diagram is shown in Fig. 4.

The first issue can arise when two consecutive estimations of Δ take the same value. In this case, the address coming from the computation of the actual difference Δ is the same as the previous one, still in the pipeline. In this case, the value is read from the same address the second time, but it is not yet updated from the previous operation. Thus, consecutive identical values of the differences will be missed from the computation of the histogram. Xilinx BRAMs have a special write mode, called WRITE_FIRST, which places the newly written data also directly on the output. This could potentially solve the issue, but this mode is not available for dual-ported BRAMs, when a port writes to and the other port reads from the same memory address, as explained in Ref. 20.

In this case, the addresses need to be compared. The value read from the BRAM the second time is ignored and one uses the updated value from the previous operation, still in the pipeline, which was not yet written back to the memory.

A slightly similar issue is when the next address is different; however, the third one is the same as the one two cycles before. In this case, the updated value is being written back to the memory, but the read value will not reflect the changes. Again, the addresses need to be compared and the updated value from the pipeline must be used instead of the one read from the memory.

In Refs. 16 and 19, these issues are not documented, although they should appear in the described corner cases, since the authors are using dual-ported BRAMs. Compared to those designs, our architecture has a longer pipeline, but it avoids the issue of two identical consecutive results for the difference Δ and is therefore correct.

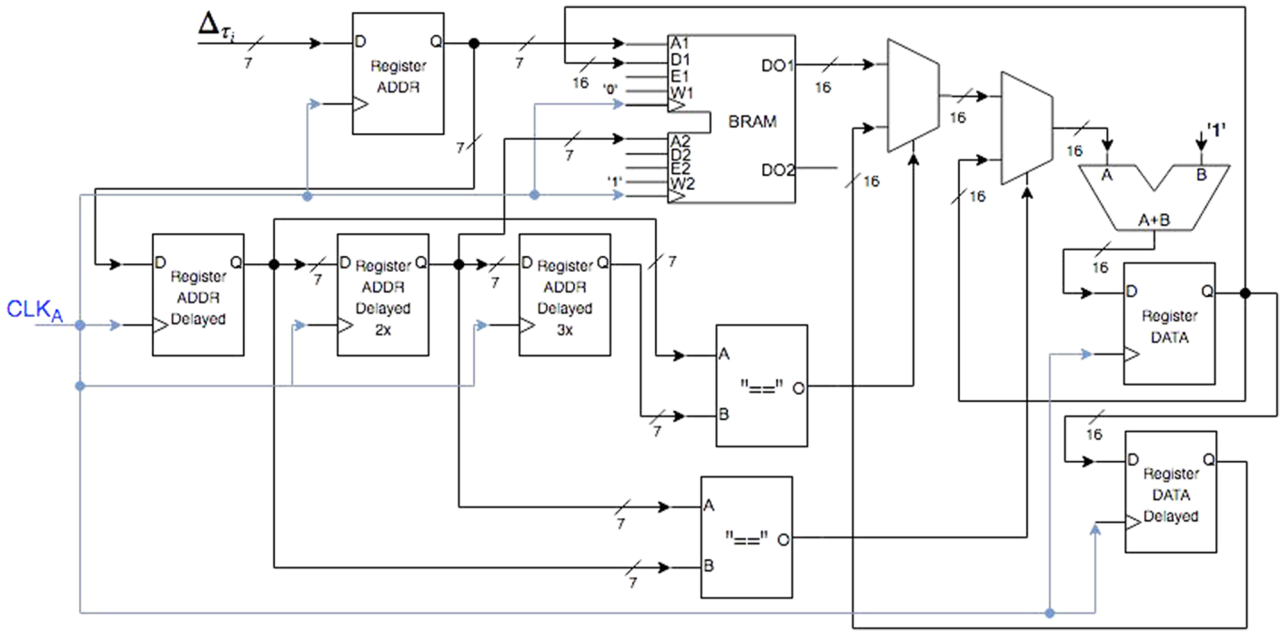


FIG. 4. Block diagram of the solution featuring a pipelined read-modify-write operation per clock cycle.

The design was optimized for speed and includes only one clock domain. Its drawback is the amount of resources that is used, since all the additional registers and other components for the pipeline must be present in all the K instances of the histogram computation blocks. As explained in the introductory paragraphs of Sec. III, minimizing the area on the FPGA chip is the top priority, so the pipeline could represent a problem from this point of view and a sequential approach would suit better.

2. Method 2 (multi-cycle read-modify-write memory operation)

Each time a new data sample is gathered, the corresponding bin of the histogram needs to be updated in one clock cycle. Since this requires multiple operations, a faster clock must be provided to this part of the design. In Ref. 17, a clock signal CLK_B that is twice faster than the data acquisition clock CLK_A is provided to the memory block. However, only 2 cycles for

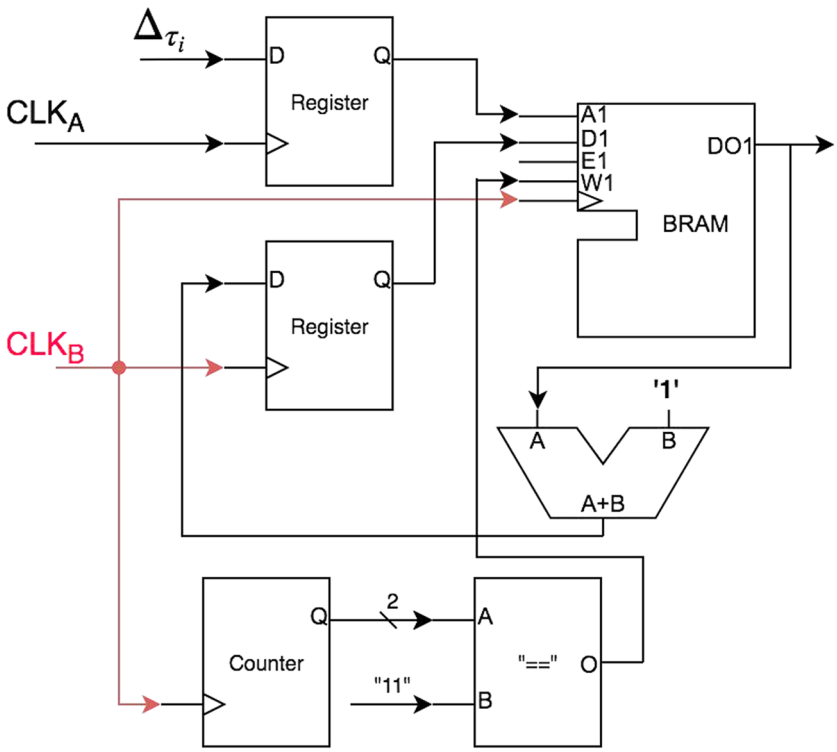


FIG. 5. Block diagram of the solution featuring a sequential read-modify-write operation using multiple clocks.

reading, updating, and writing back the value into the memory are not enough—more precisely, CLK_B must be 4 times faster than the data acquisition clock CLK_A.

In the first cycle of CLK_B, the histogram bin counter is read from the memory. In the second cycle, the read value becomes available, then it is incremented (second clock cycle), and then it gets written into a register (third clock cycle). Finally, in the fourth clock cycle, the incremented value from the register is written back to the BRAM. This procedure ensures that in the next clock cycle, the read value is actually the updated one, if the address will be the same, and it does not have the issues that were present in the previous approach. Also, an additional register is inserted between the register that stores the incremented value and the memory block to avoid any possible timing issues. Figure 5 shows the block diagram of this design, which is similar to the one from Ref. 17, but with more than two cycles of CLK_B—for the histogram update operation.

The implementation shown in Fig. 5 is optimized for area, but the design shown in Fig. 4 is optimized for speed, so it may be more useful in other contexts.

IV. EXPERIMENTAL TESTS AND RESULTS

A. Connecting with the PC

To test the design, a serial connection was established between the FPGA and a PC. This had two advantages: first, the testing data were sent from the PC to the chip using the universal asynchronous receiver-transmitter (UART) protocol at the speed of 9600 bps and this allowed to test the low-resolution data (each data sample arrived at ~100 Hz), simulating a magnetometer sensor, which also collects data at a similar rate. Using this connection, after the data were sent and the histograms created, the results were read back from the histogram BRAMs inside the FPGA chip. To verify the correctness, the normalized PDF was generated and compared to the results obtained with INA for the same range of scales τ. A second advantage is given by the protocol's simplicity.

The designs were also tested using a real magnetometer, Honeywell HMC5883L,[21] connected directly to the FPGA chip. The sensor was on a Digilent CMPS, a 3-axes digital Compass, and uses the I^2C protocol to communicate.

1. FPGA side

A Finite State Machine (FSM) was designed for the FPGA to be able to communicate with the PC/magnetometer through the serial port. So, the design needed to be extended with an additional UART interpreter component and multiple clock dividers. Of course, in the case the data samples are gathered directly by the magnetometer, the FSM only reads data samples from it and the results are sent to the PC for visualization.

The FSM has two main functionalities: it receives the data from the PC (N = 10 000 points) and it sends them to the PDF module; then, it reads the data from the specified histogram memory and sends the data back to the PC through the serial channel. Figure 6 shows the state diagram of this FSM.

To write data to the FPGA PDF module, a special *write* command must be issued, "0x01," followed by the number of samples being written, N, as two bytes to support numbers up to 65 535. Then the 16-bit N numbers must be sent to the FSM on the FPGA, which will read the data, construct the 16-bit data samples, and send them to the PDF module to compute the histogram of the *deltas*.

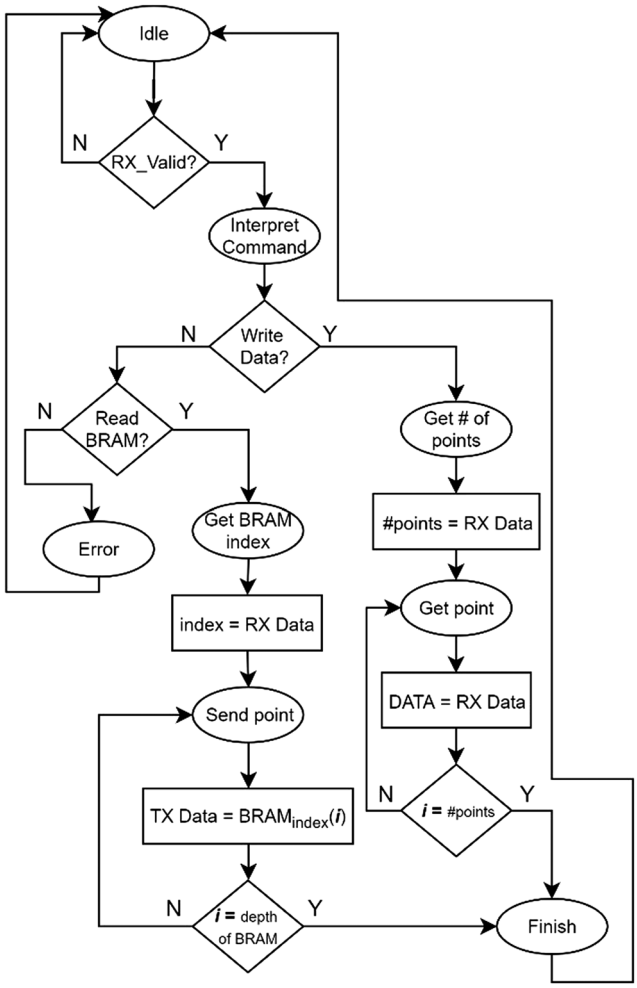


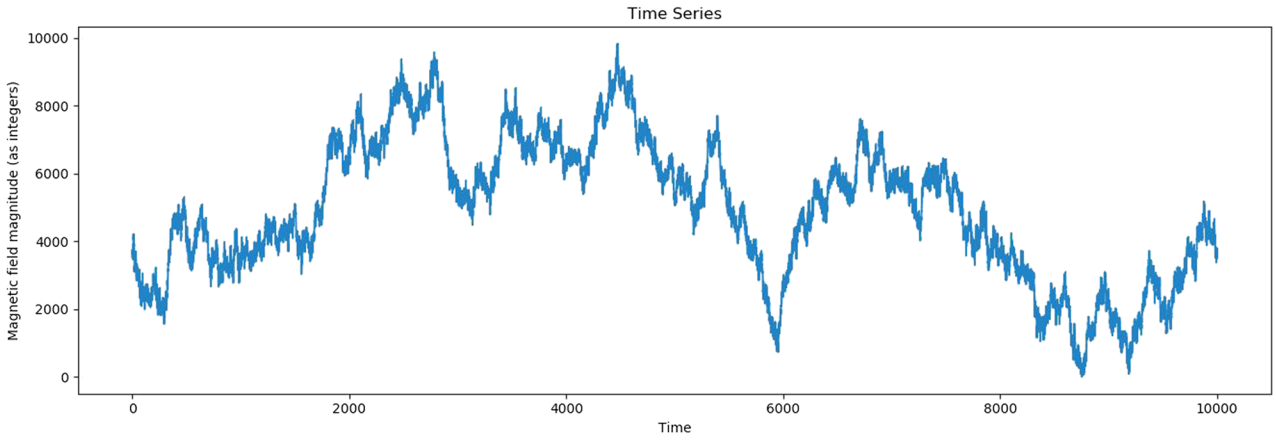
FIG. 6. State diagram of the FSM that handles the communication with the PC/magnetometer.

To read a specific histogram of a τ, the special command for read must be sent, "0x02," followed by the index of the memory to be read, corresponding to the specific τ (a value between 0 and k_{MAX} = 13). Then, the FSM will start reading the memory and send the values back to the PC.

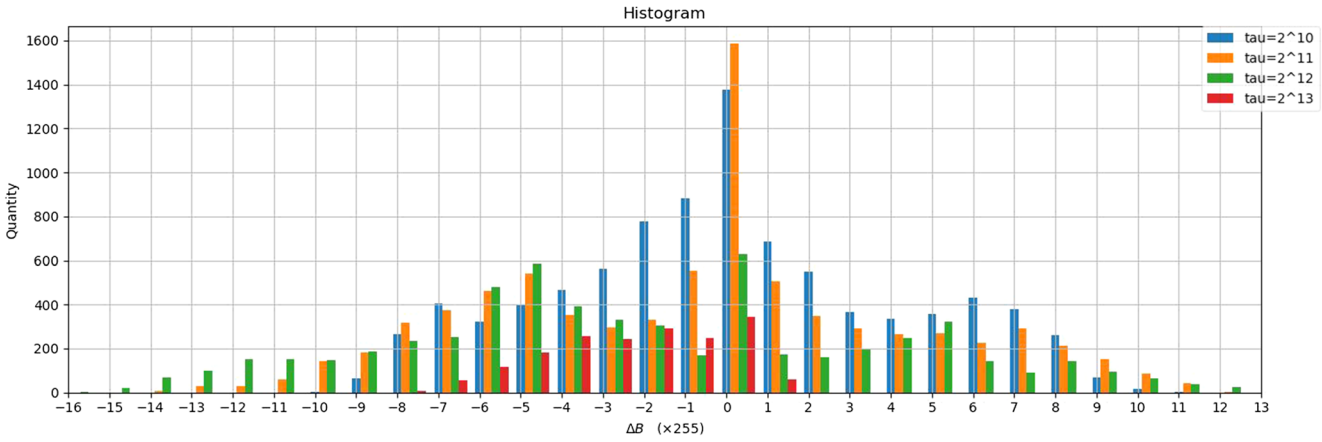
2. PC side

To communicate with the FPGA, a Python program was written on the PC side. It sends all the data to the FPGA, starts asking to read back all the histograms for each τ, computes the normalized histogram, and shows all this information on one single graph. It can also compute the PDFs directly from the data from the file, as shown in Fig. 7.

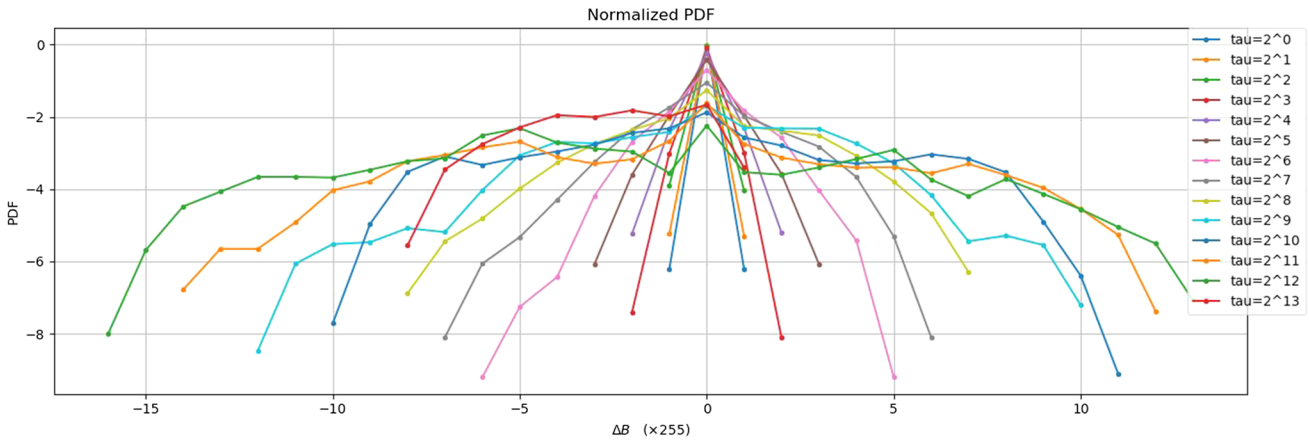
Panel (a) shows a synthetic signal similar to *in situ* magnetic field measurements. After data are processed on the FPGA with the algorithms described above, the final histogram is transferred from the device on the PC, and then it is plotted, as it can be seen in panel (b). For clarity sake, only a reduced set of histograms corresponding to four different



(a)



(b)



(c)

FIG. 7. (a) Synthetic signal similar to *in situ* magnetic field measurements; (b) histograms computed for the differences (3) applied on the synthetic signal for four different separations/scales; (c) PDFs computed for the full range of scales.

separations/scales, $\tau = 2^{10}$–2^{13} points, is shown in Fig. 7(b). The histograms corresponding to different separations/scales are shown with different colors. Panel (c) shows the color-coded probability distribution functions derived from normalized histograms computed for the full range of scales, from $\tau = 1$ to $\tau = 8192$. The shape and values of PDFs were confirmed by an independent analysis performed by means of a scientific data analysis toolbox (the Integrated Nonlinear Analysis library).

The signal depicted in Fig. 7 was created by the Matlab library INA developed by the FP7 project STORM. The dataset consists of a random walk signal, where each new sample is obtained from the previous one by randomly adding or subtracting a random number.[22]

The system is able to process any real data samples gathered from real measurements made in space (we have also used magnetic field measurements from the Cluster

spacecraft[23]), but for testing purposes, it can also use data read by an off-the-shelf magnetometer.

B. Experimental results

The experimental tests were made on a 7 series Xilinx FPGA, Artix-7 XC7A100T. Table II gives an overview of the resources needed to perform the analysis using the two methods discussed above. Since the results were obtained for the analysis of a single time-series (similar to one component of a vector variable), these values must be multiplied by 3 in the case of a full implementation (the presented architectures must be replicated for the two remaining components). There are $K = 14$ histograms in total and a BRAM block is allocated for each one of them, but since the size of these histograms is relatively small, the Xilinx optimization tool is able to map two histograms in one physical BRAM; thus, the reported BRAM usage is of only 7 such blocks.

The comparison of the design discussed in this paper with previously published solutions (Refs. 15–17) is not straightforward. Indeed, in previous studies, the resource utilization is presented in various ways. For instance, in Refs. 15 and 17, the resource utilization reports include a data storage block, which is necessary in a totally different application context, while in Refs. 15 and 16, the resource utilization reports also include some other application-specific computational blocks—thus, the comparison of the resource utilization of the full designs with the one proposed here is not entirely meaningful. Also, while the designs presented in Refs. 15–17 only compute one histogram, we compute K histograms as the application is meant to reveal multi-scale properties of signal variability.

An important constraint in space applications is the power consumption. Table III shows the power consumption of our FPGA design, as estimated by Xilinx software support tools. PDF refers to the stand-alone design (data acquisition and histogram computation blocks), while the full design refers to the whole test bench (PDF, the serial connection, and the FSM). Because only a low frequency clock signal is necessary, it will lead to a small power consumption (at least two orders of magnitude less) in comparison with a software (PC or microcontroller-based) implementation.

Note, however, that measuring the power consumed exclusively by the PDF computation module is not possible, since we are using the Nexys 4 DDR[24] development platform, which features a Xilinx Artix-7 XC7A100T FPGA chip. The problem when measuring the consumed power is that the FPGA chip is soldered on the board, which also contains additional hardware, like external memory, universal serial bus (USB), serial, and other ports, so when we measure the current drawn by the board, it also includes all the additional hardware.

One notes that using an FPGA instead of a central processing unit (CPU) or graphics processing unit (GPU) already drastically reduces the power consumption; however, even on the FPGA there may be different implementations resulting in different power requirements. One of these elements is the

TABLE II. Resource utilization for the two implementation variants (data acquisition + histogram computation modules) on an XC7A100T FPGA.

	Pipelined histogram implementation method / FPGA resources	Method 1 (pipelined read-modify-write)	Method 2 (multi-cycle read-modify-write)
Single axis design	Slice LUTs (as logic function generators)	1339 (2.11%)	1196 (1.88%)
	Slice LUTs (configured as distributed memory—SRL16/SRLC32)	4192 (22.06%)	4192 (22.06%)
	Slice flip-flops	1206 (0.95%)	385 (0.30%)
	BRAM blocks	7 (5.19%)	7 (5.19%)
	DSP48 blocks	0	0
Three axes design	Slice LUTs (as logic function generators)	4008 (6.32%)	3567 (5.62%)
	Slice LUTs (configured as distributed memory—SRL16/SRLC32)	12576 (66.18%)	12576 (66.18%)
	Slice flip-flops	3589 (2.83%)	1117 (0.88%)
	BRAM blocks	21 (15.56%)	21 (15.56%)
	DSP48 blocks	0	0

TABLE III. The power consumption of the various implementation variants.

	Pipelined histogram implementation method / FPGA power consumption	Method 1 (pipelined read-modify-write)	Method 2 (multi-cycle read-modify-write)
Single axis design	Total on-chip power (W)	0.120	0.119
	Full design (W)	0.022	0.021
	PDF (W)	0.013	0.010
Three axes design	Total on-chip power (W)	0.151	0.149
	Full design (W)	0.045	0.043
	PDF (W)	0.033	0.024

TABLE IV. Power consumption estimation of the design for different clock frequencies.

Clock frequency (MHz)	Total on-chip power (W)	Full design (W)	PDF (W)	Clock (W)
100	0.148	0.05	0.038	0.024
50	0.128	0.03	0.018	0.014
25	0.119	0.021	0.009	0.009
12.5	0.115	0.018	0.005	0.006
6.25	0.112	0.015	0.002	0.005
3.125	0.112	0.014	0.001	0.004

clock frequency: the higher it is, the more power it requires. Nevertheless, since the data acquisition rate is rather low in real space experiments, the clock frequency of the whole design can be reduced to less than 1 kHz.

The different tests made for different clock frequencies confirm that bigger clock frequencies lead to increased power consumption, as shown in Table IV.

V. CONCLUSIONS

This paper describes an FPGA-based stand-alone system suitable to be embarked on a spacecraft and to perform on-board computations of probability density functions for data recorded by scientific instruments. The tests of the FPGA solution were performed on the ground using an off-the-shelf FPGA board that analyzed synthetic as well as real datasets. The architectural solution can be easily applied to any type of vectorial data and ported to a space-grade FPGA.

Although the problem to be solved can be considered as a standalone one, it is important to remind that the logic resources of the FPGA chip will be shared by several architectures, which run several DSP algorithms. The performance criteria may thus vary according to the nature and goals of not only one, but a group of algorithms. That is why each DSP algorithm implemented in an FPGA chip built for the purpose of on-board data analysis should be designed in several variants, optimized for different criteria, area, speed, and power, to mention the most important ones.

The importance of the design presented in this paper is two-fold: (i) it computes variability descriptors that can be used by on-board or on-ground real-time data testing procedures; (ii) it allows for the analysis of data samples that otherwise would be lost due to the limited spacecraft bandwidth. At the same time, the procedure provides a reduced-dimension probabilistic descriptor of data variability, the PDFs, and thus is equivalent to a data compression algorithm.

From the technical point of view, two novel advances are reported:

1. *Acquisition and storing of the statistical ensemble of differences* for a number of K scales for a data stream of N samples: we designed a solution mainly based on look-up tables configured as SRL16/SRLC32 macros that allows a convenient indexing to compute the differences for each scale and prepare this result for direct updating of the histogram. This solution also has the advantage of saving a significant number of slices, thus decreasing the area occupied on the FPGA chip, which becomes usable for other related algorithms.
2. *Computing and updating a number of simultaneous K histograms*. Here two different approaches were tested:
   a. The *pipelined read-modify-write memory operation* method is synchronized with the data acquisition clock and achieves a maximal operational frequency. This method is appropriate for any application domain.
   b. The *multi-cycle read-modify-write memory operation* per multiple clock domains that is optimized for resources which are necessary for the implementation. This method is appropriate for space applications, where it is very important to fully exploit the logic resources available in the FPGA chip by implementing several DSP algorithms inside it.

All the design variants were tested in laboratory on the 7 series Xilinx FPGA, Artix-7 XC7A100T, with an off-the-shelf magnetometer data and synthetic data. The obtained results are meaningful, consistent with the results provided by classical nonlinear analysis routines (e.g., STORM INA). The FPGA implementation reduces the power consumption by two orders of magnitude compared to classical on-board data analysis solutions. The next step is to optimize the design for space-qualified FPGA architectures and to perform additional on-ground experiments prior to in-flight tests.

ACKNOWLEDGMENTS

This work was supported by the Romanian Space Agency (ROSA) via a grant from the Research Program for Space Technology Development and Innovation and Advanced Research (STAR Project No. 182/2017) and the UEFISCDI PCCDI project VESS (Contract No. 18/2018). M.E. also acknowledges support from the Belgian Solar-Terrestrial Center of Excellence (STCE) and thanks Dr. Mircea Ciobanu for his collaboration and support over the years.

[1]T. Dudok de Wit and V. V. Krasnoselkikh, "Non-Gaussian statistics in space plasma turbulence: Fractal properties and pitfalls," Nonlinear Proc. Geophys. **3**(4), 262–273 (1996).
[2]P. Garcia Lopez *et al.*, "Edge-centric computing: Vision and challenges," ACM SIGCOMM Comput. Commun. Rev. **45**(5), 37–42 (2015).
[3]S. A. Fuselier, W. S. Lewis, C. Schiff, R. Ergun, J. L. Burch, S. M. Petrinec, and K. J. Trattner, "Magnetospheric multiscale science mission profile and operations," Space Sci. Rev. **199**(1-4), 77–103 (2016).
[4]B. Silverman, *Density Estimation for Statistics and Data Analysis* (Chapman and Hall, London, UK, 1998).
[5]L. F. Burlaga, "Intermittent turbulence in the solar wind," J. Geophys. Res. **96**(A4), 5847–5851, https://doi.org/10.1029/91ja00087 (1991).
[6]E. Marsch and C. Y. Tu, "Non-Gaussian probability distributions of solar wind fluctuations," Ann. Geophys. **12**(12), 1127–1138 (1994).
[7]M. M. Echim, H. Lamy, and T. Chang, "Multi-point observations of intermittency in the cusp regions," Nonlinear Proc. Geophys. **14**(4), 525–534 (2007).
[8]M. B. B. Cattaneo, G. Moreno, G. Russo, and J. D. Richardson, "MHD turbulence in Saturn's magnetosheath downstream of a quasi-parallel bow shock," J. Geophys. Res. **105**(A10), 23141–23152, https://doi.org/10.1029/2000ja000093 (2000).
[9]FP7 Project STORM, European Union, 2015, Available online: http://www.storm-fp7.eu/.

[10]W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet Things J. **3**(55), 637–646 (2016).
[11]W. Yu, F. Liang, X. He, W. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," IEEE Access **6**, 6900–6919 (2018).
[12]X. S. Le, "Software/FPGA co-design for edge-computing: promoting object-oriented design," Ph.D. thesis, University of Western Brittany, Brest, France, 2017, available at https://tel.archives-ouvertes.fr/tel-01661569 and https://tel.archives-ouvertes.fr/tel-01661569/document.
[13]A. de la Piedra, B. An, and A. Touhafi, "Sensor systems based on FPGAs and their applications: A survey," Sensors **12**(9), 12235–12264 (2009).
[14]T. Kuwahara, "FPGA-based reconfigurable on-board computing systems for space applications," Ph.D. thesis, Institute of Space Systems Universitat Stuttgart, Stuttgart, 2010, available at https://d-nb.info/1000794369/34 and https://elib.uni-stuttgart.de/handle/11682/3847 and https://www.semanticscholar.org/paper/FPGA-based-Reconfigurable-On-board-Computing-for-Kuwahara/360e542bdd6ed95d3b5e2a8662ee9cb479767954.
[15]S. A. Fahmy, "Histogram-based probability density function estimation on FPGAs," in IEEE *International Conference on Field-Programmable Technology (FPT)* (IEEE, Beijing, China, 2010).
[16]E. Jamro, M. Wielgosz, and K. Wiatr, "FPGA implementation of strongly parallel histogram equalization," in *IEEE Conference on Design and Diagnostics of Electronic Circuits and Systems* (IEEE, Krakow, Poland, 2007).
[17]A. Shahbahrami, J. Y. Hur, J. Ben, and S. Wong, "FPGA implementation of parallel histogram computation," in 2nd HiPEAC Workshop on Reconfigurable Computing, Gothenborg, Sweden, 2008.
[18]See https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug953-vivado-7series-libraries.pdf for "Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide," 20 December 2017.
[19]P. Alfke, "Creative uses of block RAM," 4 June 2008, Available online: https://www.xilinx.com/support/documentation/white_papers/wp335.pdf.
[20]See https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf for "7 Series FPGAs Memory Resources," 27 September 2016.
[21]Honeywell, "3-axis digital compass IC HMC5883L," February 2013, Available online: https://aerocontent.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf.
[22]D. Sornette, *Critical Phenomena in Natural Sciences* (Springer, Berlin, 2000).
[23]A. Balogh, M. W. Dunlop, S. W. H. Cowley, D. J. Southwood, J. G. Thomlinson, K. H. Glassmeier, G. Musmann, H. Luhr, S. Buchert, M. H. Acuna, D. H. Fairfield, J. A. Slavin, W. Riedler, K. Schwingenschuh, and M. G. Kivelson, "The cluster magnetic field investigation," Space Sci. Rev. **79**(1/2), 65–91 (1997).
[24]Digilent, "Nexys 4 DDR," Digilent, Available online: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start.