Edge computing in space: Field programmable gate array-based solutions for spectral and probabilistic analysis of time series

Cite as: Rev. Sci. Instrum. 90, 114501 (2019); doi: 10.1063/1.5119231
Submitted: 9 July 2019 • Accepted: 16 October 2019 •
Published Online: 4 November 2019

Laurenţiu Opincariu,[1,a] Norbert Deak,[1,b] Octavian Creţ,[1,c] Marius Echim,[2,3,d] Costel Munteanu,[2,e] and Lucia Văcariu[1,f]

AFFILIATIONS

[1] Technical University of Cluj-Napoca, Cluj-Napoca, Romania
[2] Institute of Space Science, Măgurele, Romania
[3] Royal Belgian Institute for Space Aeronomy, Brussels, Belgium

[a] Laurentiu.Opincariu@cs.utcluj.ro
[b] Norbert.Deak@cs.utcluj.ro
[c] Octavian.Cret@cs.utcluj.ro
[d] echim@spacescience.ro
[e] costelm@spacescience.ro
[f] Lucia.Vacariu@cs.utcluj.ro

ABSTRACT

This paper addresses the problem of performing time series analysis on-board a spacecraft, where the number of constraints is much bigger than for applications running in regular (i.e., ground-based) environments. An objective of modern spacecraft technologies designed for space exploration is to perform on-board data processing tasks, in order to increase the amount of data available for scientific analysis. Field Programmable Gate Array (FPGA) devices are considered as good candidates for hardware implementations of such systems. In order to optimize the usage of on-board resources, FPGAs share their resources between several digital signal processing (DSP) algorithms. In this paper, we describe the design and implementation of such an optimized design where two DSP algorithms are implemented on the same FPGA: (1) the power spectral density and (2) the multiscale probability distribution functions. The entire implementation process is described in detail, including a discussion about the main architectural choices. The proposed solutions focus on optimization of area, speed, and power. The tests performed, on both synthetic and real data, demonstrate the feasibility of our approach and constitute the first step toward porting the design on space-grade FPGAs.

I. INTRODUCTION

The increasing performance of scientific instruments qualified for space flight is challenging for the integration of these instruments on-board spacecrafts. Moreover, the exponential growth of the amount of data recorded on-board is not matched by a corresponding increase in the capabilities to send all these data to the ground. Therefore, in order to take full benefit of the increased performance of the scientific experiments, additional efforts are needed to design and test smart devices able to perform key tasks for on-board data analysis. One key principle discussed in this paper is to cumulate on a single device/chip several functions adapted for the analysis of a specific dataset. The goal is to provide a tool able to extract a maximum amount of information from the data with minimized expenses of energy and computing resources.

The feasibility of our goals was already demonstrated in a previous paper,[1] where we developed a Field Programmable Gate Array (FPGA)-based solution to compute probability distribution functions (PDFs) of fluctuations. The solution was proposed in two implementation variants in the idea that they will be combined with other space specific applications of the same class.

Data variability is generally closely linked to the dynamical behavior of systems/environments investigated by *in situ* techniques. Our interest focuses mainly on variability of space plasma parameters, such as the magnetic field, the bulk speed, and the density, measured *in situ* in the solar wind and/or the terrestrial magnetosphere. These parameters help characterizing the solar-terrestrial interactions and space weather phenomena. Several important data variability descriptors are provided by time series analysis (TSA) or digital signal processing (DSP) techniques.

The Power Spectral Density (PSD) reveals how the energy is distributed over the frequency range by decomposing the signal in eigenvectors represented by trigonometric functions. This representation is valid under the assumption of stationarity. The multiscale Probability Density Functions (PDFs) describe the statistics of fluctuations and do not rely on a specific representation of mathematical functions. However, its accuracy depends heavily on the number of data samples. The description of data provided by the two methods is complementary, although PSD is seen as a low order analysis compared to PDFs. Both methods provide data variability descriptors under the form of a reduced dimension set, the PSD and the PDF, compared to the original raw data series. Thus, these methods can be seen, in a broader sense, as a data compression tool. Therefore, such an on-board data analysis tool is crucial when the local resources preclude sending the entire dataset to the ground. This is why the paradigm of edge computing becomes relevant for satellite data analysis.

Edge computing is an emerging paradigm that becomes more and more used in the scientific community. According to Ref. 2, edge computing is a method for optimization of computing systems "by taking the control of computing applications, data, and services away from some central nodes to the other logical extreme (the «edge») of the network of interconnected computing systems." In the context of space applications, this means that the control is transferred from the ground center to the computers on-board the spacecraft.

In Ref. 3, it is shown that the main benefits of edge computing come from the optimization objectives that can be reached easier than by using classical computing paradigms, especially the delay and execution time, energy and power, and scalability and availability.

Edge computing can be successfully applied in a wide variety of fields. In our context, end-to-end delay, communication cost, and data loss are the most important performance criteria. The concept of *in situ processing* can bring many solutions to improve performance. It is used in various fields such as big data,[4] memory systems,[5] fog computing,[6] and high-performance computing.[7]

The idea of edge computing is straightforward: to "push" computations onto the satellite end and send to the ground only a synthesis of the data processed in the outer space that describes however key dynamic features of the raw data. As a result, a series of advantages arise:

- the on-chip hardware will be lighter, which is a key element in the context of space applications;
- the total power consumption for performing the computations will be orders of magnitude less than in the case of a classical microprocessor-based implementation;
- the total power consumption for transmitting data to the ground will be significantly reduced, since much less data are transmitted;
- the size of the physical system will be significantly smaller, making it possible to be embarked in small satellites such as CubeSats.

As our concern is plasma survey, there are a few DSP algorithms relevant for the analysis of data samples gathered by various sensors (e.g., magnetometers, variometers, and wave analyzers). Due to high performance or efficiency requirements, these analyzers are often implemented in hardware and it is preferable that an FPGA chip placed on a satellite performs as many analyses as possible. Moreover, recent space plasma missions use FPGAs as the main technology for the electronic blocks and computing processing units that serve the scientific instruments.[8,9]

The question is: is it possible to run those DSP algorithms in FPGA chips instead of microprocessors? This paper presents solutions for implementing at least two DSP algorithms [probability distribution function (PDF) and power spectral density (PSD)] in the same FPGA chip and discusses the main architectural choices that make this possible.

Space applications need to be run on qualified hardware that is radiation-tolerant. From the developer's point of view, the logic structure is identical on a regular off-the-shelf FPGA chip and on a space-grade one. Nevertheless, the logic components are physically triplicated in a space tolerant FPGA, in a way that is transparent for the developer and that increases the reliability. The space radiation unexpected events span a large range, like highly energetic particle impacts on the chip, that can alter the value of parts of the design, like bits stored in memory, programmable interconnection points (PIPs), etc.

We tested our solution in a laboratory, on regular FPGA chips, as a first step toward prototyping a space-qualified solution. The main novelty at this step is the conceptual and algorithmic design of computations typical for the algorithms involved. The solution can later on be ported to space-qualified chips.

The rest of this paper is organized as follows: Sec. II presents a theoretical background of the two DSP algorithms mentioned above. It also includes a survey of previous work done in this field. The proposed architectural solutions are presented in Sec. III, while Sec. IV presents the corresponding experimental results and measurements. Conclusions and future work are presented in Sec. V.

II. BACKGROUND AND PREVIOUS WORK

A. Power spectral density

Spectral analysis is one of the most popular approaches to investigate data variability. It uses the decomposition in trigonometric (eigen)functions provided the signal $x(t)$ is stationary,

$$X_k = \sum_{j=0}^{N-1} x_j \exp\left(-i\frac{2\pi jk}{N}\right)\Delta t. \tag{1}$$

Equation (1) provides the expression for the Fourier component X_k at frequency $f_k = \frac{k}{N\Delta t}$ of a signal, x_j, discretized in N samples separated by the same time interval Δt (uniform sampling). It is a tool that proved its usefulness in many fields of science.

The technique became affordable and widely used with the introduction of the Fast Fourier Transform (FFT) algorithm, which can be implemented in various types of data processing units. FFT is currently designed in virtually all programing languages and used by scientists and engineers.

We are interested in the implementation of an algorithm that estimates the power spectral density to be implemented in an FPGA device. A power density function is the distribution of variance (mean square value) of a time series over frequency.[10] In practice, one needs to estimate the mean square Fourier component for an infinitesimal frequency interval. Several strategies can be employed to estimate the PSD with the Fourier approach; here, we use the simplest one, the periodogram,

$$S(n) = \frac{2\Delta t}{N}|X_k|^2, k = 0, \dots, \frac{N+1}{2}, \tag{2}$$

where X_k is determined with an FFT algorithm following (1). The frequency interval, $[0, \frac{1}{2}\Delta t]$, is broken into $N/2$ parts such that the PSD frequency resolution is $\Delta f = \frac{1}{N\Delta t}$. Although the estimation of the PSD with (2) is rather coarse, it satisfies the scope of our design, i.e., a robust and rapid procedure to estimate the spectral properties of the signal when rather limited computing resources are available. We do not apply any windowing procedure prior to FFT, nor do we average several PSDs as in the Welch algorithm to reduce the PSD noise (see Ref. 11). The main task we assume here is to include in the same FPGA device the algorithms able to compute simultaneously estimations of the multiscale PDFs and PSD.

Fourier analysis is widely applied in space data analysis for time series, images, etc. Although FPGAs have increasingly prominent roles in the design of electronics blocks of state-of-the-art plasma missions (see, e.g., Ref. 12), to our knowledge, there is no such FPGA designed to perform PSD estimations on-board the spacecraft.

Note also that data gaps are known to generate many undesirable side effects when estimating the spectral properties of a signal (see, e.g., Ref. 13). Thus, the final design should include additional logic to tackle these effects. This problem is outside the scope of the present paper, and it will not be discussed further.

B. Probability distribution functions

The concept of the probability distribution function is fundamental for the statistical description of data variability and, in a more general sense, for understanding the dynamics of complex systems such as space plasmas.[14] It offers a complete image of the statistics of "events" from small amplitudes to extremes, provided the number of samples is large enough such that the "wings" of the PDFs are reasonably well sampled. Indeed, the later give a quantitative measure on the occurrence rate of "rare" or "extreme" events (see, e.g., Ref. 15). The definition of an "event" is of course depending on the quantitative measure associated with the relevant physical quantity (e.g., velocity in the case of neutral fluid turbulence and magnetic field for space plasma turbulence). In our studies, we adopt a differential measure, as explained below, and shall check its scaling properties.

An accurate estimation of PDFs from experimental data is not straightforward, as discussed by Ref. 16. Several algorithms are available; here, we adopt the one that is easier to implement due to its relatively reduced mathematical complexity—the histogram based method. This approach was applied in several studies of space plasma variability from *in situ* data (e.g., Refs. 17–20).

The mathematical kernel for the computation of the multiscale PDFs is available from a data analysis toolbox, the Integrated Nonlinear Analysis (INA) library.[21] INA is built to serve as a scientific investigation tool, adapted to the needs of scientists and controlled by a graphical user interface. INA includes several nonlinear data analysis tools and algorithms from lower to higher order methods.

Let us consider a generic time series $x(t)$. To address its fluctuating characteristics, we may form the scale-dependent difference measure as the time series,

$$\delta x = x(t+\tau) - x(t). \tag{3}$$

We therefore consider the normalized histograms of differences δx at scale τ to be an estimation of the probability distribution functions (PDFs), $P(\delta x, \tau)$, for a range of temporal scales τ (see technical details on the histogram methods in our previous paper[1]). The FPGA-based implementation of the histogram algorithm for computing multiscale PDFs follows closely the approach adopted in INA.

C. Previous FPGA-based PSD implementations

In Ref. 22, the authors presented a fixed point implementation for a PSD estimation system based on B-spline windows. The system processes frames of only 1024 input samples represented using two's complement fractional numbering system (which is a fixed-point format), where both the input samples and the generated results have a 24-bit precision. The authors do not present the way they implemented the FFT block, and the implementation was done on a Virtex5 FPGA.

In Ref. 23, the authors presented a model-based design approach and implementation for the calculation of the PSD of the received signal inside the spectrum scanner of a cognitive radio system. The signal processing comprises the implementation of a control unit, selectable window functions, the FFT, the calculation of the magnitude, the selectable number of averages for the calculation of peak and mean power values, as well as the logarithmic representation of the spectrum. The PSD calculation has been realized with a system generator, which is a model-based design approach from Xilinx using MATLAB/Simulink as a development environment, and it has been carried out to operate on the targeted FPGA within the MicroBlaze processor system. The Xilinx IP core is used for FFT calculation over 512 samples with no dedicated windowed step before, while for magnitude computation, the CORDIC algorithm was used.

In Ref. 24, the authors presented an FPGA based design for power spectrum analysis. Their platform allows for the real-time data acquisition and processing of samples of the incoming signal in a small time frame and sustains simultaneous data streams on each of the four input channels. The processing consists of computation of power and its average and peak, over a set of input values. Using reverse engineering techniques, they rely on the power spectrum analysis in order to reveal information such as frequency and source about any interference, which impacts and changes the characteristics of a signal. FFT computation is done over 256 fixed point input values using the FFT CoreGen component from Xilinx, which was considered convenient for multichannel processing.

In Ref. 25, the authors proposed an implementation of a digital phosphor trigger. Their system transforms input time-domain signals into frequency domain, by means of high-speed FFT, and then plots the signals' spectrum into a bitmap, which can be used by the system, based on classification techniques to trigger different judgments. LogiCORE IP Fast Fourier Transform v7.1 was used for FFT calculation over 1024 points, and the square and logarithmic operations in the actual power calculation are based on the CORDIC IP core.

Although the FPGA designs are increasingly used in space applications, it seems that none of the above-mentioned FPGA solutions is appropriate for the specific context of our targeted applications, which is characterized by a significant complexity and the need of the design to be stand-alone.

The most important difficulty of such designs that target an FPGA implementation is to collect/store a sufficient number of samples on the chip for relevant analysis, until it becomes possible to run the algorithms. Thus, the area occupation becomes the most important critical factor, since the data sampling rate is much lower than the operational frequency of the FPGA chip.

D. Previous FPGA-based PDF implementations

The idea of implementing PDF estimators in FPGA chips is not new, as the PDF is a widely used tool in many types of applications and data analyses, one of the most important being particle physics.[26]

A step-by-step PDF estimator, whose results are available in real time, is presented in Ref. 27. The author discussed the design optimization criteria and concluded, like we do in this paper, that the area is the most important one. In the design presented in Ref. 27, in order to save the logic resources of the FPGA chip, the histogram is stored in dual-ported Block Random Accessed Memories (BRAMs).

The authors of Ref. 28 also use multiple dual-port BRAMs to store histograms. The goal is to perform parallel histogram equalization for images. Each image pixel is evaluated in a single clock cycle: the histogram bin is read, incremented, and written back into the BRAM block. Since dual-port BRAM blocks are used, read and write operations can be done in parallel, thus increasing the speed.

In Ref. 29, the authors proposed an FPGA-based solution for computing the Gaussian copula PDF. The authors considered speed to be the main performance criterion, and the proposed design aims at optimizing it. The proposed architecture is not stand-alone: it is designed as a hardware accelerator inside a general-purpose computing system and it is pipelined.

The idea from Ref. 28 is also discussed in Refs. 30 and 31. It presents the same approach of updating a value, i.e., performing a read-modify-write operation in one clock cycle (port A of the BRAM is used as a read port, while port B is used as a write port), and uses a common clock signal.

The stand-alone system presented in Ref. 1 proposes a few FPGA-based solutions for computing multiscale PDFs, the main optimization goal being the minimal amount of logic resources. It also demonstrates the reduction of the power consumption by two orders of magnitude compared to classical on-board data analysis solutions.

III. PROPOSED FPGA-BASED PDF AND PSD IMPLEMENTATIONS

The solutions we propose in this paper for PSD and PDF calculation are fully stand-alone and optimized for the logic area, as they must both fit in the same FPGA chip. The entire hardware system can be integrated into the central processor unit of the spacecraft and able to analyze data in real-time. It can be seen as the base for a bigger system which eventually can compress and extract a series of key-parameters qualified to describe the variability of physical quantities measured *in situ* by scientific instruments.

The hardware support for the implementations presented here is a Xilinx FPGA chip, which usually contains four major types of hardware resources: *logic slices*—used for implementing Boolean functions, memory blocks (called "distributed memory"), or small FIFOs/shift registers; *slice Flip-Flops*—containing up to 8 Flip-Flops; *BRAMs*—18/36 kbit memory blocks with dual port capabilities; *DSP blocks*—which can implement basic multiply and accumulate operations, and can be clocked at twice the frequency of the rest of the logic resources.

A. PSD implementation

Power spectrum analysis based on FFT implementations became classic in the last few decades. Given the targeted environment of our application and space, where tremendous amount of limitations are imposed, most of the optimization effort is oriented toward minimizing the area occupied on the FPGA chip.

We work with the scenario that the data to be analyzed on-board is vectorial (e.g., provided by a three-axis magnetometer). In comparison with other implementations where in general only a limited number of input samples are processed in a data frame, we consider that a minimum number of $N = 8192$ sample points are necessary to compute the PSD for each magnetic field component; these data need to be stored on the FPGA chip.

The algorithm to be implemented is described below (Algorithm 1). The block diagram of the whole system is shown in Fig. 1.

1. FFT computation

In the context of space applications, it is essential to have the best possible computational precision, while minimizing the space occupied in the chip. Below, we present the main tools available to perform the PSD calculation, their advantages, and disadvantages.

ALGORITHM 1. PSD computation of data variance.

```
1. For each $t_i$ with $i$ from 0 to $N$
2.     Read measured data sample $A(t_i)$
3.     Save it in the corresponding resource
5.         Compute $B = FFT(A)$
6.         Store $B$ in the corresponding resource
7.         Compute $C = PSD(B)$
8.         Store $C$ in the corresponding resource
9. End for
```

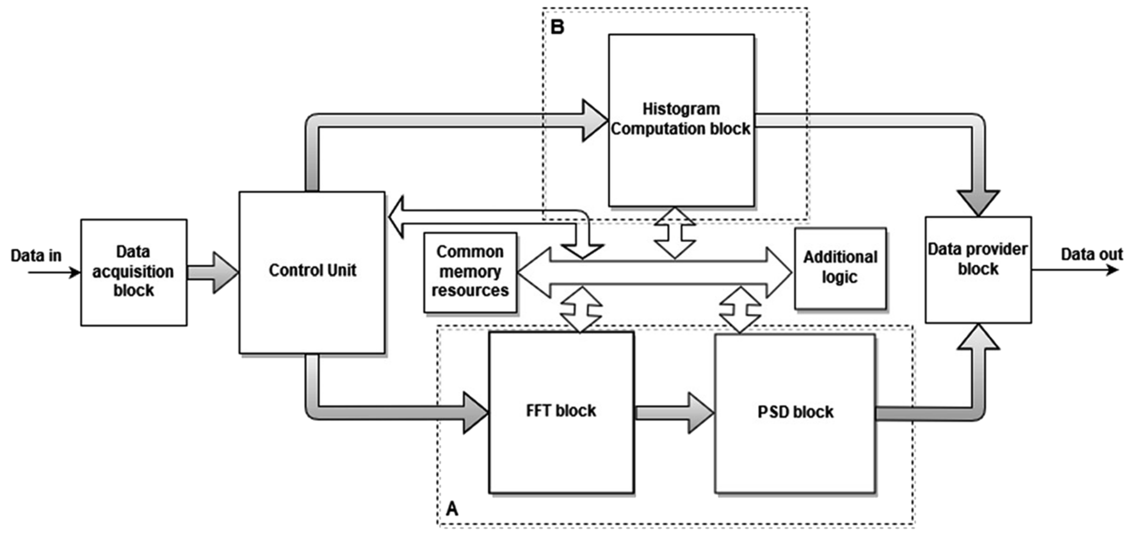


FIG. 1. The block diagram of the system where module A is in charge with PSD computation and module B computes the histograms and PDFs.

The problem of implementing FFTs in FPGAs is solved in various contexts, and many IP cores are available for the development of both scientific and industrial applications. A few IP cores were developed for floating-point operators needed for our application.

The objective is to perform FFT calculation for N spectral components represented in single-precision floating-point—see Eqs. (1) and (2). Below, we provide a comparative analysis of the IP cores discussed in the literature for PSD/FFT computation.

2. Xilinx DFT (discrete Fourier transform) LOGICORE PG106

Xilinx provides several IP cores for their FPGA families. This IP core[32] implements forward and inverse DFTs for a wide range of user-selectable point sizes. The bit-width of the input data samples may vary from 8 to 18, and the numbers are expressed in two's complement numbering system. Xilinx also delivers a bit-accurate C model to support software simulation.

This IP core is easy to use, does not require a huge amount of resources, but the data array size is not a power of two, which requires additional logic on the chip. This IP core does not support FFT computation over more than 1536 data samples, which was insufficient in our context.

The input is represented as a complex two's complement fixed-point value, while the output is a complex block floating-point value, as defined for the forward transform.

3. VIVADO IP CORE PG 109

This IP core[33] implements the DFT based on the classical Cooley-Tukey FFT algorithm.[34] The number of input data samples must be a power of two (but less than 65 537). It is capable of generating the FFT for both fixed point and single-precision floating-point inputs and can be configured to process FFT on up to 12 different channels.

This IP core is based on the AXI Interface. The IP core configuration menu available in VIVADO gives the developer the possibility to choose the parameters for the IP core.

This IP core provides run-time configurable transform length, and thus the FFT size can be dynamically modified. However, the streaming width cannot be configured, which means that the core can only be fed with one complex point in a transaction, thus leading to scalability limitations.

4. SPIRAL

The SPIRAL DFT/FFT IP generator[35] automatically generates customized DFT modules in synthesizable Verilog. Like in most IP core systems, there is a variety of parameters that are user customizable, such as transform size, transform direction, data type, architecture, radix, streaming width, data ordering, and BRAM budget.

Two types of architectures need to be taken into consideration when computing FFT: the *fully streaming architecture*, which allows data to stream in and out of the system continuously, and the *iterative architecture*, which consists of a single stage.[35]

In order to be able to integrate the Verilog modules generated by SPIRAL in our project, only a few minor modifications were needed. Along with the BRAM budget (option that allows the user to specify the maximum number of BRAMs to use when targeting a Xilinx FPGA), SPIRAL gives the user the opportunity to set the streaming width and radix in order to tailor the IP core to the application's needs.

We considered that the iterative architecture is the best candidate to be used for this spectral analysis.

In conclusion, we considered that SPIRAL is the most appropriate tool for developing our application, since it is the most flexible, easy to use, provides accurate results, and allows the minimization of the on-chip area.

Table I summarizes the main characteristics of these IP cores for FFT computation.

TABLE I. Comparison between IP cores for FFT computation.

IP cores for FFT computation	Bitwidth of the input data	Inputs: Fixed-point/ floating-point	Bit-accurate C model to support software simulation	Maximum number of data samples	No. of channels	Flexibility
Xilinx DFT LOGICORE PG106	8–18	Fixed-point only	YES	1 536	1	Low
VIVADO IP CORE PG 109	8–34	Both (fixed-point OR floating-point)	YES	65 536	12	Medium
SPIRAL	4–32	Both (fixed-point OR floating-point)	NO	32 768	1	High

5. PSD estimate (periodogram) computation

The periodogram [Eqs. (1) and (2)] is computed over $N/2$ complex points [Fourier coefficients obtained above with Eqs. (1) and (2)].

6. VIVADO IP CORES Floating-Point Operator v7.1 (LogiCORE IP PG060)

The Xilinx Floating-Point Operator[36] core integrated into VIVADO allows a range of floating-point arithmetic operations to be performed on FPGA. The operation is specified at the core generation time, and each operation variant has a common interface.

As stated in Ref. 7, this IP core "complies with much of the IEEE-754 standard. The deviations generally provide a better trade-off of resources against functionality." However, the PSD estimates computed using these cores were sometimes different from the ones generated using the scientific software INA.

7. FLOPOCO

FloPoCo (an acronym for "Floating-Point Cores") is an open-source framework for the generation of arithmetic datapaths developed in C++ by de Dinechin *et al.*[37]

FloPoCo operators are designed in the idea of "computing just right:" no bit is computed if it is not useful, which means that the user can choose various parameters in order to generate only the necessary amount of logic resources in the FPGA chip for implementing the target application. This framework makes the assumption that by designing operators that are fully parameterized in precision, one may obtain more accurate results with less hardware and in less time than the arithmetic units of general purpose computer systems.

In order to use the generated operators the user has to adjust the input and output format. More precisely, the user has to transform floating point inputs into the format required by FloPoCo and the results back again in IEEE 754 single-precision binary floating-point format (or other format used).

The user can select a target FPGA chip, but the latest Xilinx families are not supported yet (the last family available was Virtex5). However, FloPoCo is in a continuous development, and updates are expected to appear for all the operators.

In terms of used resources, the differences between the two IP cores are not significant, but FloPoCo infers more DSP48E1 blocks. However, the biggest problem is the inaccuracy of the results obtained with the VIVADO IP core when compared to INA results, which led us to choosing FloPoCo for the implementation of this block of our architecture.

8. Implementation of the global PSD block

The goal was to implement the PSD estimator for as many samples as possible with maximal precision and minimizing the area (amount of occupied resources) in the FPGA chip. The architecture was tailored to process 8192 samples, but in order to compare with previously reported implementations, we also present the results for 1024 samples.

As mentioned before, after performing tests with all the IP cores, alone and in all the possible combinations, we decided that the best combination for our application is composed of SPIRAL (for the FFT computation) and FloPoCo (for the floating-point operations).

The block diagram of this architecture is shown in Fig. 2. X is a parameter that depends on the problem's size, $X = \log_2(N) - 1$.

The control unit is a Finite State Machine (FSM) that controls the entire system. Data samples are first received byte by byte and stored iteratively in the BRAMs until the entire data frame is formed.

Once that the complex values are received, the FFT block is notified using the "data_to_process_ready_fft" signal (Fig. 2) which will trigger the FFT computation. The FFT block is actually a wrapper over the SPIRAL core, which is fed with complex values read from BRAMs. This block collects and stores Fourier coefficients back in BRAMs, thus iteratively replacing data samples that were used for the linear transform. Once the FFT computation is done, the control unit is notified, which at its turn starts the PSD computation through the "data_to_process_ready_ps" signal (Fig. 2).

The PSD block was generated with FloPoCo. In the PSD block, the FFT coefficients are read from BRAMs and PSD coefficients are written in the same BRAMs, thus replacing the previously obtained FFT coefficients. After the PSD estimate computation is done, the control unit reads PSD data from memory and sends them to the PC.

We have used 4 independent BRAMs instead of a single one to store data samples, FFT coefficients, and PSD coefficients for the following reasons. Our power spectral density tool is a generic architecture developed in such a way that it can be used for a concrete implementation by using any of the IP cores available for the FFT and PSD computation. We had to divide the needed memory in four BRAM blocks because of the module generated by SPIRAL, which,

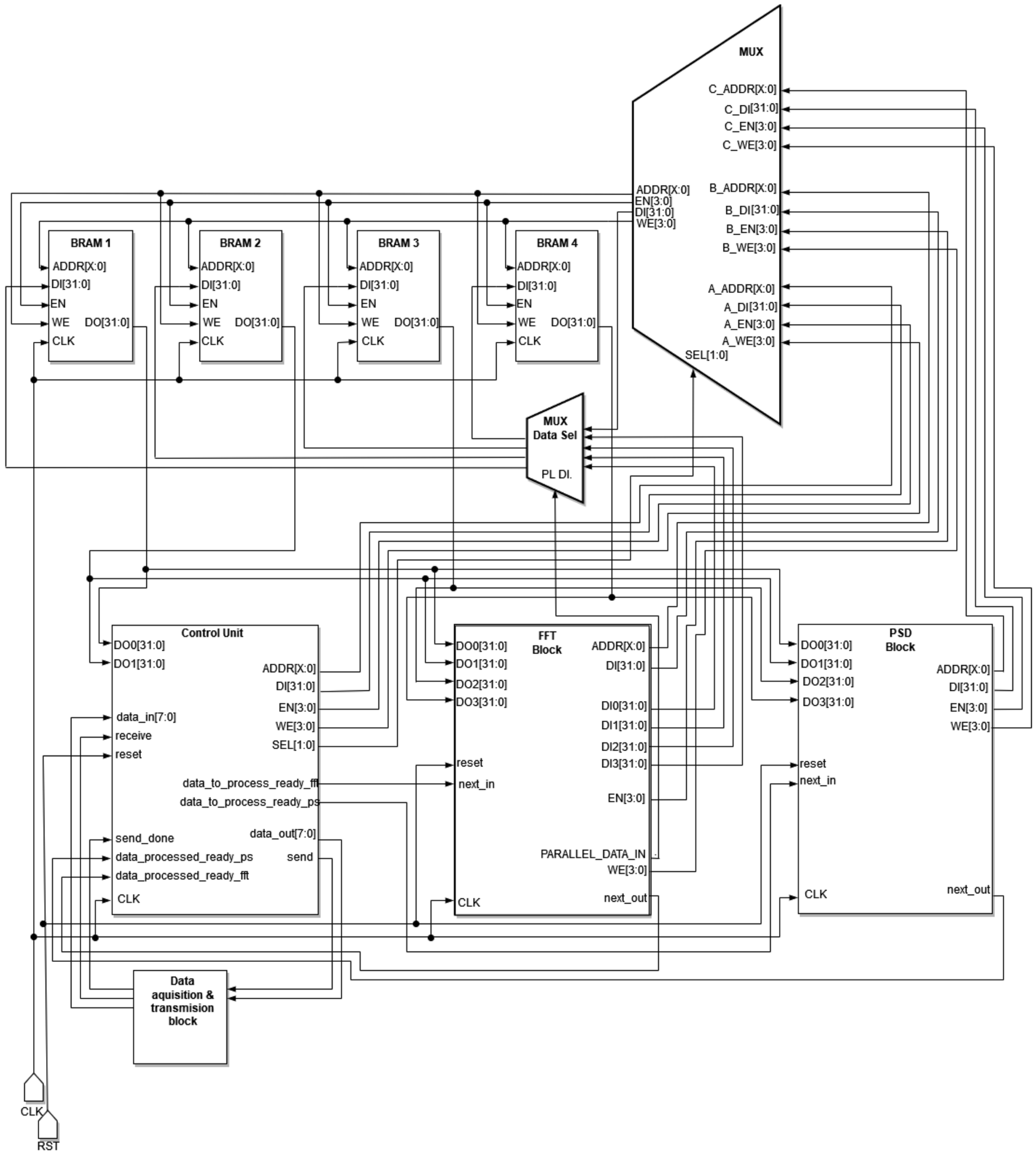


FIG. 2. Block diagram of the PSD computation module.

configured with a streaming width of two, expects as inputs two real and two imaginary values on each clock. It also yields as outputs two FFT coefficients on each clock cycle, until the transform is done.

Reading from BRAMs into an auxiliary array in order to feed the FFT block with 4 values in a clock cycle and to collect results into an array was not a feasible idea because when *N* is big, the synthesis tool will try to infer BRAMs. Accessing more than 2 elements in the same array on the same clock cycle will either cause the synthesis tool to replicate the BRAM, so more read ports would be necessary, or if the synthesis tool cannot figure that out, it will instead build the array from loose slice Flip-Flops. If the array is large like in our case, this would indeed cause long synthesis time and moreover the number of control sets needed might exceed device capacity.

The main aspect is the fact that this architecture based on four BRAMs is able to fetch in one clock cycle both operators needed to feed the PSD computation module. This is possible because the read/write operations are implemented by means of two separate BRAMs. The final results of the PSD computation module are

ALGORITHM 2. PDF computation.

```
1. For each $t_i$ with $i$ from 0 to $N$
2.     Read measured data sample $A(t_i)$
3.     Save it in the corresponding resource
4.     For each $\tau$
5.         Compute $A(t_i) - A(t_i - \tau)$
6.         Increment the corresponding bin
7.     End for
8. End for
```

residing in only one BRAM. An architecture based on a single BRAM would have a more complex, which would have had consumed more valuable logic resources that can be used for implementing other algorithms on the same FPGA chip.

Basically, we divided the memory in four BRAMs, each one of them storing $N/4$ data samples to make sure that the 4 values needed in every clock cycle are available in parallel. Thus, it became possible to both feed the FFT block and store its results directly from and into all the BRAMs in parallel by using a Multiplexer which allows writing different values, both sequentially and in parallel, in each BRAM.

B. PDF implementation

The architectural solution for implementing PDF computation is based on a simple algorithm (Algorithm 2), which consists of two main parts: data acquisition and histogram computation, as shown in Fig. 1.

Since the input data sample is common to the PSD and PDF modules, it uses the values already saved in the BRAMs instead of replicating the input data on the FPGA by storing them somewhere on the chip.

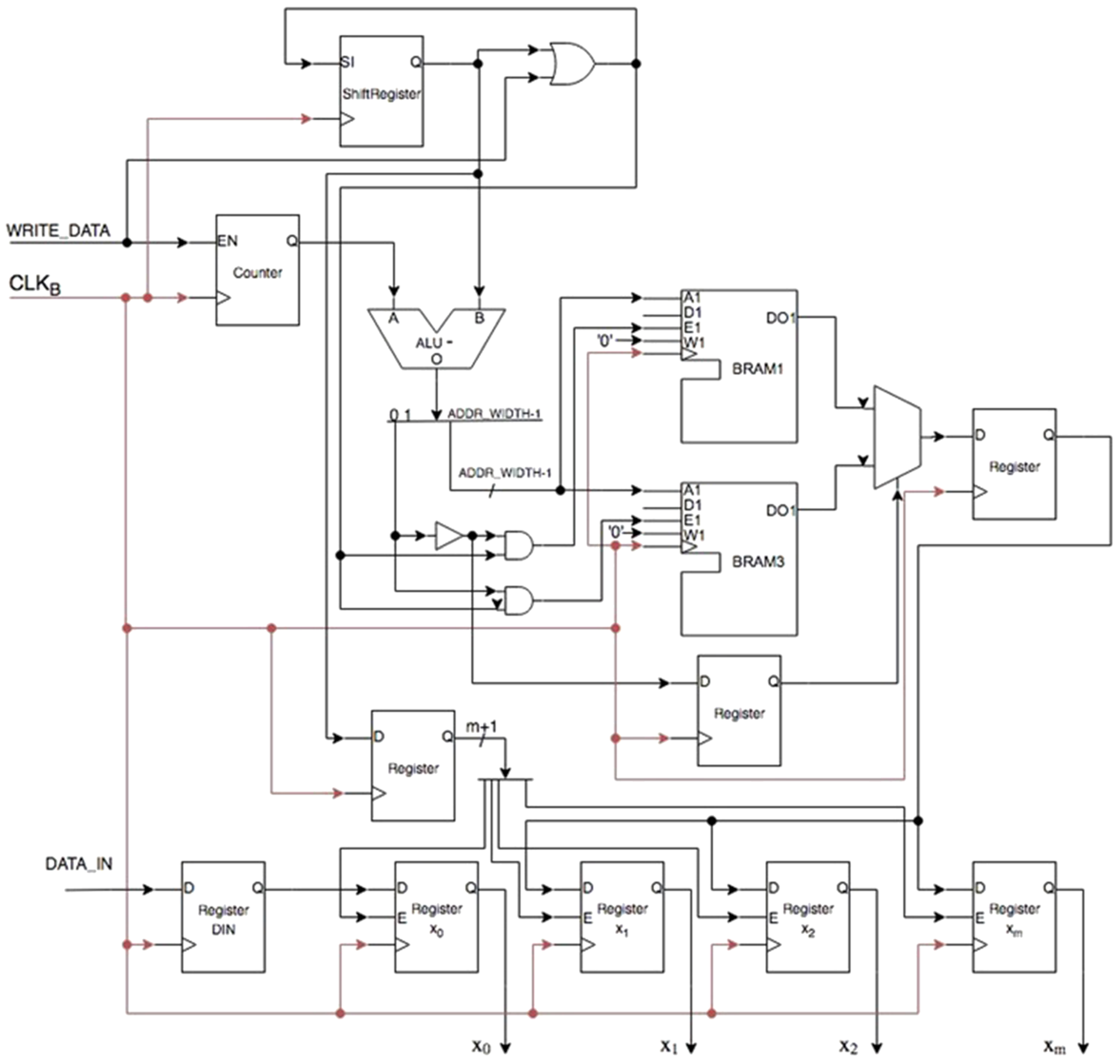


FIG. 3. Block diagram of the BRAM-based architectural solution for histogram computation in the PDF module.

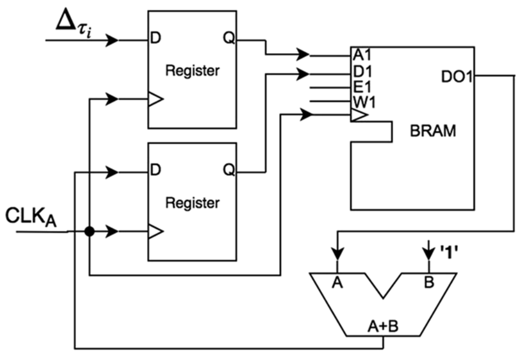


FIG. 4. Histogram computation block for one τ.

1. Data acquisition block

Here, we only need the data samples at τ (the temporal scale) distance from the currently gathered data sample (see Sec. II B); thus, it is enough to be able to access them only at the corresponding indices. When a new data sample is read at a specific clock rate (CLK_A), K other data samples need to be read from the memory, until the arrival of the next data sample to compute the $\Delta_\tau = \boldsymbol{A}(t) - \boldsymbol{A}(t - \tau)$ differences for each τ in parallel (lines 4–7 from Algorithm 2) and update the corresponding bin for each histogram. After the new data sample is written to the BRAM block at the corresponding address, the values from τ distance must be read sequentially from the BRAM (K previously stored data samples). All these operations must be performed until the next data acquisition cycle is finished. That is why it is necessary to ensure a clock signal that must be at least K times faster than the data acquisition clock. We denote this signal by CLK_B. Each time a data sample is read, the difference (Δ) is computed and the result is stored in a separate register (one for each τ). Before a new sample arrives, each data sample located at the specific distance from the new data sample is subtracted from the latter, the result being available in a separate register.

To compute the indices of the data samples that must be read, a shift register is used, as shown in Fig. 3 and discussed in Ref. 1. At the arrival of a new sample, a "1" is written to the end of the shift register, and it is propagated through the shift register at the CLK_B rate. Thus, in each "small" cycle of CLK_B, we get a binary number that represents a power of which is subtracted from the current memory address given by the counter.

The biggest advantage of this design is the small amount of resources used, especially memory (this represents only $1/K$ of the solution where each data BRAM would have been duplicated K times to obtain full parallelism); and since it uses the BRAMs from the PSD implementation module, the full circuit requires half of the memory resources for storing the input data, compared to duplicating them and saving them in a separate BRAM specifically for the PDF circuit.

2. Histogram computation block

This design must compute $K = 14$ different histograms in parallel, i.e., the histogram of the results from $x_1{-}x_0$, $x_2{-}x_0$, $x_4{-}x_0$, $x_8{-}x_0$, . . ., $x_{8192}{-}x_0$. The width of each histogram bin is user customizable, but the design is much simpler if this number is a power of 2. Since the input data samples are 16-bit numbers, the differences will have the same bit-width. If we choose to have, for instance, $128 = 2^7$ bins for each input data sample, the 7 most significant bits of the result have to be pruned out, which will constitute the index of the corresponding bin.

The histogram is composed by the resulting collection of K bins, where each bin contains the number of *deltas* (differences) in a given interval. The best way to store these numbers is in BRAMs (the same solution was adopted in Refs. 1, 27, 28, and 38). Each time a new Δ is computed, its corresponding bin is incremented using a scheme like the one presented in Fig. 4.

The PDF algorithm implies computing a different histogram for each τ, so the schematic from Fig. 4 is replicated K times. Each histogram bin has a maximal capacity of 256 × 16 = 4096 bits because we can have at most 256 bins. Since the capacity of a Xilinx BRAM is

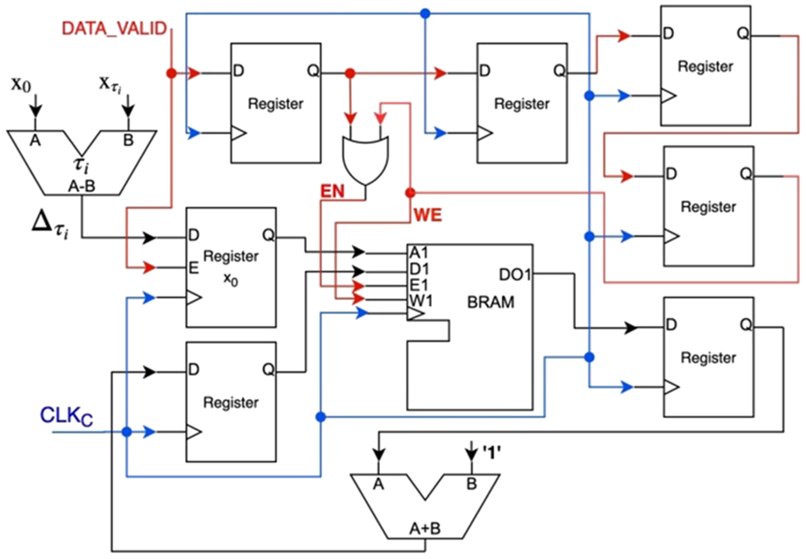


FIG. 5. Block diagram of the solution featuring a sequential read-modify-write operation.

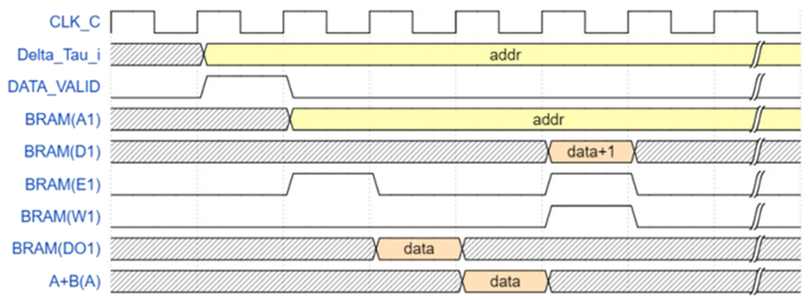


FIG. 6. The waveform of the sequential read-modify-write operation.

36 kbits, the histogram bin will surely fit in one BRAM, which gives a total of K BRAMs needed in this design.

Incrementing a value from a memory location comprises three elementary steps: first, the value needs to be read out, then incremented, and then written back into the memory (this is a read-modify-write memory operation). All these operations should theoretically be performed in one clock cycle. However, since these operations deal with the same memory location, each operation of reading and writing from and to the BRAM requires at least one clock cycle—so, in order to handle this problem, it is mandatory to design an additional schematic, like the one presented in Fig. 5.

With every new data sample that is gathered, the corresponding bin needs to be updated in one clock cycle. Since this requires multiple operations, a solution is to use pipelining with a faster clock CLK_C, whose frequency must be at least 4 times greater than the data acquisition clock frequency, CLK_A (this is similar to the usage of CLK_B, as explained in Sec. III B 1), because it needs four clock cycles to correctly update the histogram.

In the first two cycles of the faster clock, CLK_C, the value of the histogram bin counter is read from the memory and saved to a register. Then, the value is incremented and written into another register. Finally, the incremented value is written back to the BRAM, while the enable signal is also propagated to this stage of the pipeline, so it will also serve as *Write Enable* (WE). In addition, the address of the histogram bin is saved in a register, to be available when the incremented value is written back. Each time a new set of data is available from the data acquisition block, a *DATA_VALID* enable signal becomes "1" for one CLK_C cycle. This protocol is presented as a waveform diagram in Fig. 6.

Since each difference delta (Δ) becomes available in a CLK_B cycle (Sec. III B 1), which is K times faster than CLK_A, it is convenient if CLK_C has the same frequency as CLK_B, so the update of the K histograms can begin right after each difference is obtained.

This solution differs from the one presented in Ref. 1 for both the data acquisition block and the histogram computation block. In Ref. 1, the input data were stored in one large shift register, using only LUTs, with accesses to the data at various indices in order to compute the K different histograms. The reason for choosing a different solution was to save the BRAMs for PSD processing algorithms and to make sure that in each clock cycle, when a new data sample arrives, the K other data samples needed for the Δ differences are directly accessible. This increases the maximal operational frequency of the design. On the other hand, the data samples are the same as for the PSD computation, so there is no need to store them separately, thus saving a great amount of resources per global. However, in order to access the K other data samples, a faster clock is used, which limits the operational frequency.

Overall, the histogram computation method from this paper may achieve a slightly lower operational frequency compared to the one in Ref. 1, but it uses fewer resources, especially Slice LUTs.

IV. EXPERIMENTAL TESTS AND RESULTS

A serial protocol was developed to establish a communication between the FPGA chip and a PC. The benefit of this approach is that the testing data are sent from the PC to the FPGA chip at the transfer rate of 9600 bps, which is quite close to the rate at which data samples are gathered by a magnetometer sensor (~100 Hz). This connection is used to simulate data gathering from an external sensor. After the input data are ingested, they are processed by the two on-chip algorithms described above. To verify the correctness of the on-chip processing, the PSD and the normalized PDF results are compared with the ones obtained with the scientific software INA.[21]

The designs were also tested using a real magnetometer, Honeywell HMC5883L,[39] which was connected directly to the FPGA chip. The sensor was on a Digilent CMPS, a 3-axes digital compass, and it uses the I^2C protocol to communicate. This device is designed for simple compassing and magnetometry applications. Its maximal field resolution is of 2 mG in ±8 G fields, obtaining a 1°–2° degrees compass accuracy. It has low energy consumption. Its digital interface and 160 Hz maximum output rate are appropriate for testing our designs in this context.

A. Connecting with the PC

A Finite State Machine (FSM) was designed to establish the communication between the FPGA chip and the PC and/or the magnetometer through the serial port. Hence, the design needed to be extended with an additional UART interpreter component and multiple clock dividers. In case 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; in case data samples are read from a file from the PC (simulated data), the communication with the PC will be bidirectional.

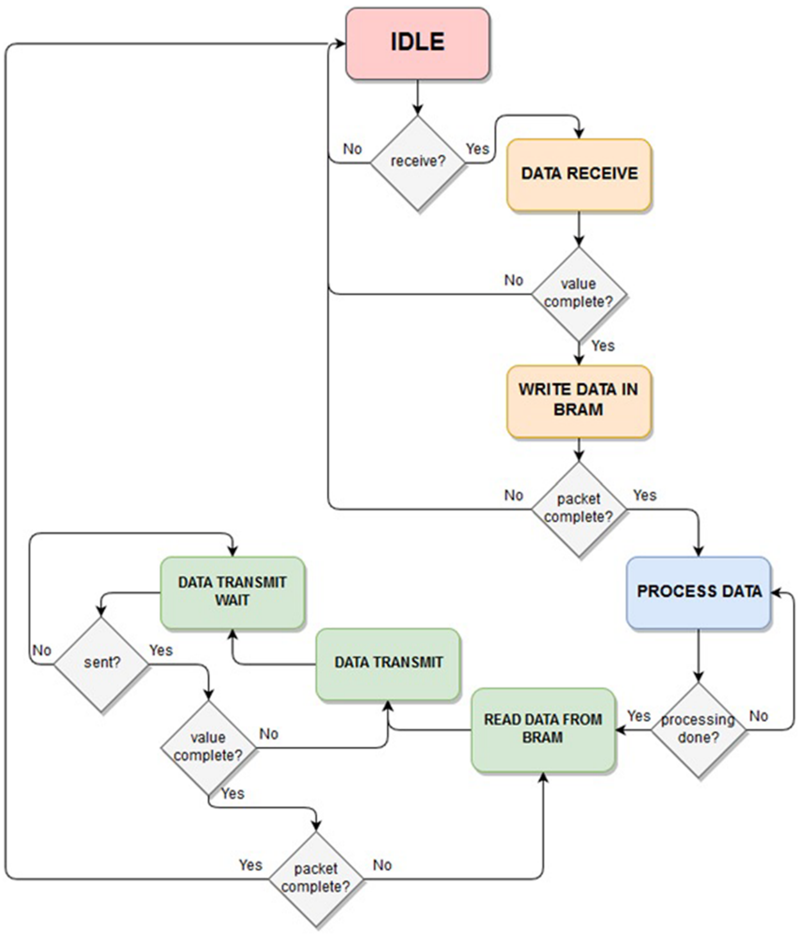
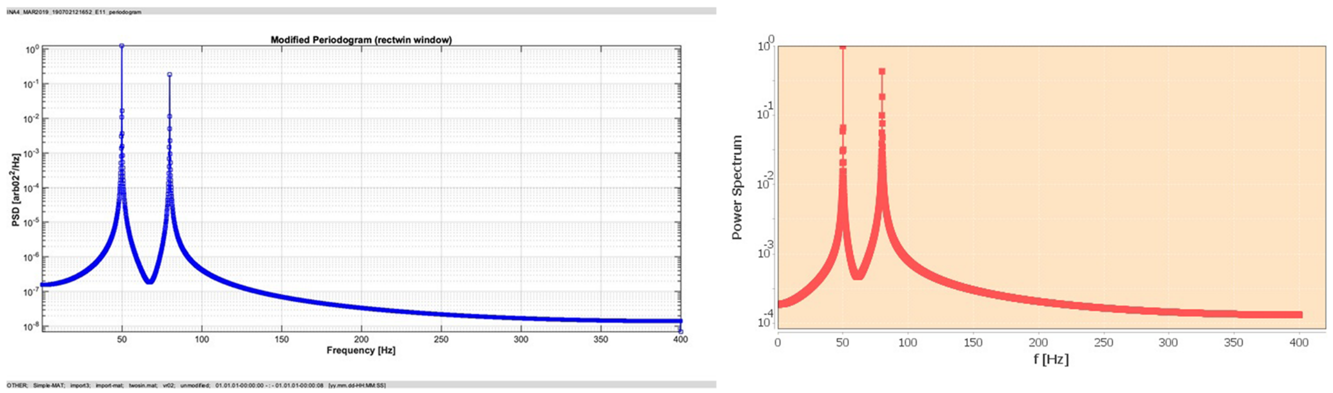


FIG. 7. State diagram of the FSM that handles the communication with the PC/magnetometer.

The FSM receives the data from the PC/magnetometer ($N = 8192$ points) and sends them to the FPGA computation blocks. At the end of the computation process, the FSM reads the results from the memory and sends them back to the PC through the serial channel. Figure 7 shows the state diagram of this FSM.

To communicate with the FPGA, a Java program was written on the PC side. It sends all the data to the FPGA, waits for the results, and shows all this information on graphs.

This system is able to process real data samples gathered from measurements made in space, but in order to confirm the



(a) (b)

FIG. 8. Power spectrum estimates for a synthetic signal of 8192 data points (a superposition of two sine waves with frequencies of 50 and 80 Hz): (a) graphic generated using INA and (b) graphic generated using the on-chip results.

TABLE II. Chip occupation report for the PSD design.

Data samples (N)	Slice LUTs (used/total)	Slice registers (used/total)	BRAM tiles (used/total)	DSPs (used/total)
1024	5055/63 400 (7.97%)	6402/126 800 (5.05%)	12/135 (8.89%)	10/240 (4.17%)
2048	5047/63 400 (7.96%)	6446/126 800 (5.09%)	40/135 (17.78%)	10/240 (4.17%)
4096	5055/63 400 (7.97%)	6490/126 800 (5.12%)	68/135 (35.56%)	10/240 (4.17%)
8192	5039/63 400 (7.95%)	6535/126 800 (5.15%)	96/135 (71.11%)	10/240 (4.17%)

TABLE III. Power consumption—estimated for the final PSD architecture with 1024, 2048, 4096, and 8192 data samples, respectively.

No. of input data samples	Total on-chip power (W)	FFT computation (W)	PSD estimate computation(W)
1024	0.292	0.151	0.038
2048	0.388	0.243	0.039
4096	0.484	0.335	0.040
8192	0.581	0.429	0.041

accuracy, we used custom synthetic signals created using the INA library developed within the FP7 project STORM.[21] To validate this project even more, we also used a magnetometer as the data source.

B. On-chip results

1. PSD implementation results

The experiments were made on a 7 series Xilinx FPGA, Artix-7 XC7A100T.

Figure 8 presents a comparison between spectral analysis provided by INA software[21] and our FPGA solution, for 8192 data points. The input signal was a superposition of two sinus functions with frequencies: 50 and 80 Hz, respectively. The results show good accuracy confirming that the FPGA solution performs well and provides a good estimation of the PSD of the test signal.

One can see from Table II that depending on the number of data samples considered for the analysis only the BRAM amount increases proportionally with N. The other columns in Table II show that the other aspects do not suffer from any modifications. In addition, the processing time required for a data frame is directly proportional to N, the total number of samples. As stated before, the architecture can process both real and complex values, which basically doubled the amount of memory needed for storage. In the event of the implementation of this solution in space applications, by simple modifications of a few indexes, the space required by the architecture can be considerably reduced by preparing it for real signal processing only. In addition, since the results from Table II are for only one axis of the magnetometer, these values must be multiplied by 3 in the case of a full implementation.

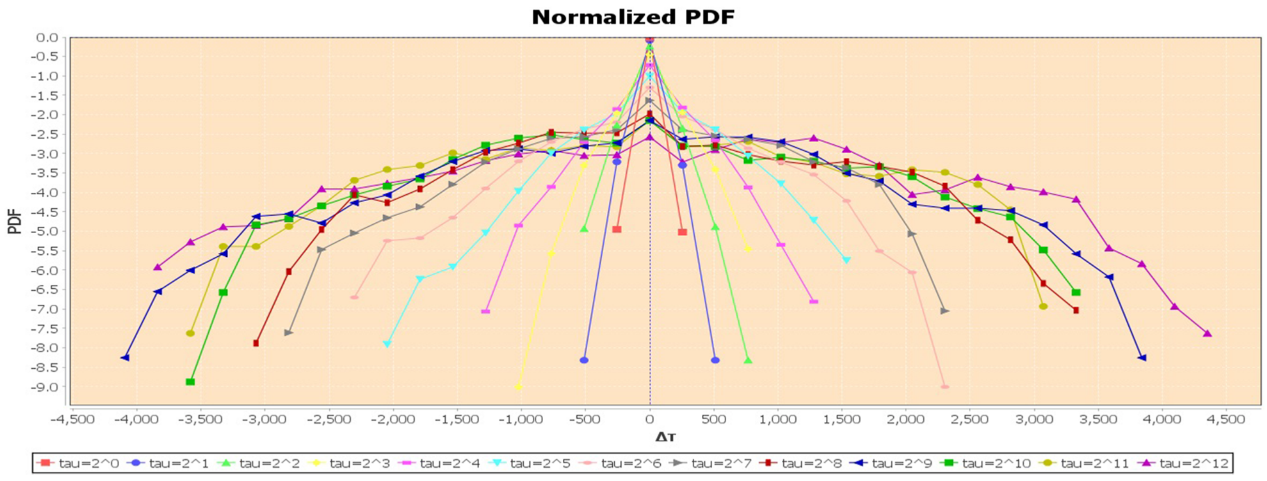
As a conclusion, we can state that the main objective of our design, data compression, is indeed accomplished. This FPGA architecture achieves a 50% data compression rate for real signals, whereas in the case of complex signals, the compression increases to 75%. These solutions may therefore increase the scientific output of space exploration missions.

TABLE IV. Resource utilization report for the PDF computation module.

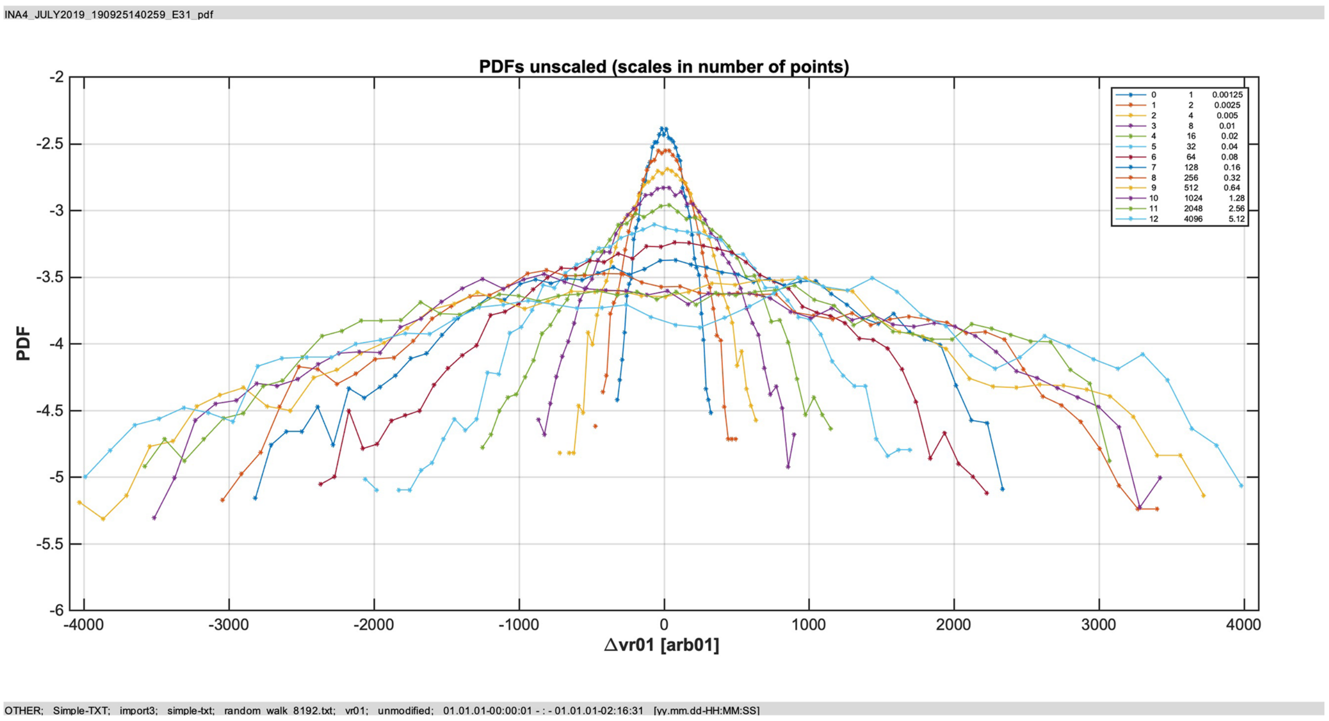
Data samples (N)	Slice LUTs (used/total)	Slice registers (used/total)	BRAM tiles (used/total)	DSPs (used/total)
1024	1016/63 400 (1.6%)	691/126 800 (0.54%)	5/135 (3.7%)	0/240 (0%)
2048	1102/63 400 (1.74%)	745/126 800 (0.59%)	5.5/135 (4.07%)	0/240 (0%)
4096	1144/63 400 (1.8%)	821/126 800 (0.65%)	6/135 (4.44%)	0/240 (0%)
8192	1218/63 400 (1.92%)	886/126 800 (0.7%)	6.5/135 (4.81%)	0/240 (0%)

TABLE V. Resource utilization of the full design, containing both PSD and PDF.

Data samples (N)	Slice LUTs (used/total)	Slice registers (used/total)	BRAM tiles (used/total)	DSPs (used/total)
1024	6267/63 400 (9.88%)	7198/126 800 (5.68%)	17/135 (12.59%)	10/240 (4.17%)
2048	6329/63 400 (9.98%)	7309/126 800 (5.76%)	45.5/135 (33.33%)	10/240 (4.17%)
4096	6392/63 400 (10.08%)	7420/126 800 (5.85%)	74/135 (54.81%)	10/240 (4.17%)
8192	6455/63 400 (10.18%)	7531/126 800 (5.94%)	102.5/135 (75.93%)	10/240 (4.17%)



(a)



(b)

FIG. 9. The normalized PDF histogram for 8192 data points: (a) results of the FPGA implementation and (b) results of the software implementation.

Another important aspect to be considered when designing solutions for the space field is power consumption. Table III shows the power consumption of the FPGA device, as estimated by Xilinx software support tools. 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.

2. PDF implementation results

Table IV shows the resource utilization by the PDF computation module for different input set sizes. These values include mostly the histogram computation part, and the data storage part, without the BRAMs since these blocks are shared with the PSD computation module.

Table IV also shows that increasing the input data sample size (and, implicitly, the value of K) leads to a higher amount of resources used. This is reflected especially in BRAM tiles, since one τ requires one additional histogram, which is contained within a half BRAM block.

Table V presents the resource utilization of the full design for the FPGA, including both the PSD and PDF implementations, for different sizes of the input data. There is a tiny change in the amount of logic resources, no change at all for DSPs; however, the BRAM blocks are used intensively. Therefore, it is important to reduce the amount of memory resources, by sharing the BRAMs between PSD and PDF.

Figure 9 shows a normalized PDF histogram for 8192 data points, for a random walk signal: (a) in the case of the FPGA implementation (the normalization was done on the PC, after the histogram was read back from the FPGA) and (b) in the case of the software implementation using INA.

3. Latency and throughput

For the FFT computation, the latency is 57 279 clock cycles; as for the throughput, a new transform is done every 57 279 cycles. As for the PSD estimation, the latency is 114 688 clock cycles; as for the throughput, a new transform is done every 53 275 cycles (585 data samples are computed/s).

Since the PDF computation module uses three clocks with different frequencies, computing the latency is more complicated. The data is sampled at the frequency of CLK_A; CLK_B is used for computing each Δ_τ from the sequence, so the frequency of CLK_B is K times faster than the frequency of CLK_A. Finally, the histogram is updated using CLK_C, whose frequency is at least 4 times faster than the frequency of CLK_A, but for convenience, it has the same frequency as CLK_B. Hence, the latency of the PDF computation block is $K + 4$ using CLK_B because after the last Δ_τ is computed, the histogram is updated in 4 additional clock cycles.

The throughput of the PDF computation module is equal to the data sampling, which is 9600 bps, i.e., 300 data samples/s.

These values exclude the data acquisition/transmission time.

4. Power consumption

Table VI presents the power consumption of the full design, estimated by Xilinx tools. The PSD computation module is largely dominant, because of the intensive computations that occur in the PSD circuit. The BRAM blocks that store the input data samples are also considered as part of the PSD circuit, while for the PDF, simple arithmetic operations (addition and subtraction) and a few memory read-write operations are performed. Still, the total power consumption is two orders of magnitude lower than on PCs or microprocessors.

TABLE VI. Power consumption—estimated for the final design, including both PSD and PDF circuits.

Data samples	Total on-chip power (W)	PSD (W)	PDF (W)	Histogram (W)
1024	0.306	0.189	0.013	0.007
2048	0.404	0.282	0.013	0.006
4096	0.502	0.376	0.013	0.004
8192	0.600	0.470	0.013	0.003

V. CONCLUSIONS

In this paper, we propose a single chip, FPGA-based architecture for efficient on-board analysis of space data using DSP algorithms as PSD and PDF.

Two important aspects were addressed: area and accuracy. Both algorithms are synthesized on a single FPGA and process in real-time up to 8192 data samples on an Artix7 technology. The computations' accuracy was confirmed by comparing the results with the ones produced by a certificated scientific library adapted specifically for space data analysis.

One of the biggest advantages of the proposed architecture is that it can be very easily scaled depending on the targeted device. In addition, the power consumption of this design is up to two orders of magnitude less than a processor-based implementation, an aspect that is facilitated by the relatively low rate at which the data acquisition process is done in a spatial context.

Given the space application context, the effective weight of the whole system is also crucial, and from this point of view, a single-chip FPGA-based implementation like the one proposed in this paper is an almost ideal solution.

All the important aspects of such a design in the spatial context were presented together with simulation results, which demonstrate the potential of this solution to be integrated in the space technologies on-board modern scientific spacecraft. This paper shows how logic resources can be effectively shared in the FPGA chip to obtain two hardware implementations of some fundamental DSP algorithms for space applications, with all the benefits brought by a hardware implementation, in terms of power consumption, area, weight, and speed.

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). M.E. and C.M. acknowledge support from the Romanian Ministry of Research through Programme NUCLEU.

REFERENCES

[1] N. Deak, O. Creţ, M. Echim, E. Teodorescu, C. Negrea, L. Văcariu, C. Munteanu, and A. Hângan, "Edge computing for space applications: Field programmable gate array-based implementation of multiscale probability distribution functions," Rev. Sci. Instrum. **89**(12), 125005 (2018).

[2] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," ACM SIGCOMM Comput. Commun. **45**(5), 37–42 (2015).

[3] C. Li, Y. Xue, J. Wang, W. Zhang, and T. Li, "Edge-oriented computing paradigms: A survey on architecture design and system management," ACM Comput. Surv. **51**(2), 39:1–39:34 (2018).

[4] A. Javed Awan, M. Ohara, E. Ayguade, K. Ishizaki, M. Brorsson, and V. Vlassov, "Identifying the potential of near data processing for apache spark," in Proceedings of the International Symposium on Memory Systems (MEMSYS '17), Alexandria, Virginia, USA, 2017.

[5] P. Siegl, R. Buchty, and M. Berekovic, "Data-centric computing frontiers: A survey on processing-in-memory," in Proceedings of the Second International Symposium on Memory Systems (MEMSYS '16), Alexandria, VA, USA, 2016.

[6]M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," IEEE Commun. Surv. Tutorials **20**(3), 1826–1857 (2018).
[7]E. Vermij, C. Hagleitner, L. Fiorin, R. Jongerius, J. van Lunteren, and K. Bertels, "An architecture for near-data processing systems," in *Proceedings of the ACM International Conference on Computing Frontiers (CF '16)* (ACM, New York, 2016).
[8]E. Taylor, P. Harvey, M. Ludlam, P. Berg, R. Abiad, and D. Gordon, "Data processing unit for THEMIS," Space Sci. Rev. **141**, 153–169 (2008).
[9]R. E. Ergun, S. Tucker, J. Westfall, K. A. Goodrich, D. M. Malaspina, D. Summers, J. Wallace, M. Karlsson, J. Mack *et al.*, "The axial double probe and fields signal processing for the MMS mission," Space Sci. Rev. **199**(1-4), 167–188 (2016).
[10]K. Otnes and L. Enochson, *Applied Time Series* (Wiley-Interscience, New York, 1978).
[11]G. Paschmann and P. W. Daly, *Analysis Methods for Multi-Spacecraft Data* (International Space Science Institute, Bern, Switzerland, 1998).
[12]C. T. Russell, B. J. Anderson, W. Baumjohann, K. R. Bromund, D. Dearborn *et al.*, "The magnetospheric multiscale magnetometers," Space Sci. Rev. **199**(1-4), 189–256 (2016).
[13]C. Munteanu, C. Negrea, M. Echim, and K. Mursula, "Effect of data gaps: Comparison of different spectral analysis methods," Ann. Geophys. **34**, 437–449 (2016).
[14]T. T. S. Chang, *Introduction to Space Plasma Complexity* (Cambridge University Press, Cambridge, 2015).
[15]D. Sornette, *Critical Phenomena in Natural Sciences* (Springer, Berlin, 2000).
[16]B. Silverman, *Density Estimation for Statistics and Data Analysis* (Chapman & Hall, London, UK, 1998).
[17]L. F. Burlaga, "Intermittent turbulence in the solar wind," J. Geophys. Res. **96**(A4), 5847–5851, https://doi.org/10.1029/91ja00087 (1991).
[18]E. Marsch and C. Y. Tu, "Non-Gaussian probability distributions of solar wind fluctuations," Ann. Geophys. **12**(12), 1127–1138 (1994).
[19]M. M. Echim, H. Lamy, and T. Chang, "Multi-point observations of intermittency in the cusp regions," Nonlinear Processes Geophys. **14**(4), 525–534 (2007).
[20]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).
[21] "FP7 Project STORM," European Union, 2015, available online at: http://storm-fp7.eu/index.php/data-analysis-tools.
[22]C. Stanciu, L. Stanciu, and V. Stanciu, "FPGA implementation for power spectral estimation using grouped B-spline windows," in *Proceedings of IEEE International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania* (IEEE, 2014).
[23]S. Riess, J. Brendel, and G. Fischer, "Model-based implementation for the calculation of power spectral density in an FPGA system," in *7th International Conference on Signal Processing and Communication Systems (ICSPCS)* (Carrara, VIC, 2013).
[24]Y. Abhyankar, C. Sajish, Y. Agarwal, C. R. Subrahmanya, and P. Prasad, "High performance power spectrum analysis using a FPGA based reconfigurable computing platform," in *IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig 2006), San Luis Potosi* (IEEE, 2006).
[25]L. Bang-qiang, W. Qian, Y. Xiao, and H. Ling, "A new spectrum measurement trigging method and its implementation," in International Conference on Computational Problem-Solving (ICCP), Jiuzhai, China, 2013.
[26]L. Lista, "Probability distribution functions," in *Statistical Methods for Data Analysis in Particle Physics* (Springer, 2016), pp. 21–51.
[27]S. A. Fahmy, "Histogram-based probability density function estimation on FPGAs," in *IEEE International Conference on Field-Programmable Technology (FPT), Beijing, China* (IEEE, 2010).
[28]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, Krakow, Poland* (IEEE, 2007).
[29]H. Ruan, X. Huang, H. Fu, and G. Yang, "A fully pipelined probability density function engine for Gaussian Copula model," Tsinghua Sci. Technol. **19**(2), 194–202 (2014).
[30]"Vivado design suite 7 series FPGA and zynq-7000 all programmable SoC libraries guide," 20 December 2017, available online at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug953-vivado-7series-libraries.pdf.
[31]P. Alfke, "Creative uses of block RAM," 4 June 2008, available online at: https://www.xilinx.com/support/documentation/white_papers/wp335.pdf.
[32]"Discrete Fourier transform v4.0, LogiCORE IP product guide, vivado design suite PG106," 18 November 2015, available online at: https://www.xilinx.com/support/documentation/ip_documentation/dft/v4_0/pg106-dft.pdf.
[33] "Fast Fourier transform v9.0, LogiCORE IP product guide, vivado design suite PG109," 4 October 2017, available online at: https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf.
[34]J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. **19**(90), 297–301 (1965).
[35]P. Milder, F. Franchetti, J. C. Hoe, and M. Puschel, "Computer generation of hardware for linear digital signal processing transforms," ACM Trans. Des. Autom. Electron. Syst. **17**(2), 15:1–15:33 (2012).
[36]"Floating-point operator v7.1, LogiCORE IP product guide, vivado design suite PG060," 4 October 2017, available online at: https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_1/pg060-floating-point.pdf.
[37]F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," IEEE Design Test Comput. **28**(4), 18–27 (2011).
[38]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.
[39]Honeywell, "3-axis digital compass IC HMC5883L," February 2013, available online at: https://aerocontent.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf.