ISSN: 1983 7402 ITA, 27 a 29 SET 2022

FPGA-Based Implementation of a Multichannel FFT Parallel Processor

Canisio Barth[1], Edgard Cansio[1]
[1]Navy Research Institute, Rio de Janeiro/RJ – Brasil

***Abstract*— A flexible and scalable architecture for a multichannel fast Fourier transform (FFT) processor, to be implemented on field-programmable gate arrays (FPGAs), is proposed in this paper. The main purpose is to evaluate the feasibility of a design that uses multiple FFT intellectual properties (IPs) to process small portions of a larger signal. Additionally, this work aims to explore the parallelism and high-speed processing intrinsic to an FPGA in the context of electronic warfare applications. Therefore, this study intends to provide a solution capable of straightly interact with the high throughput originated by radio frequency front-ends, and quickly supply frequency domain information of the incoming signals to the detection and estimation units. The proposed architecture was implemented and tested on a *Zybo Z7-20 development board*. The experiments show that the parallel bank of FFT IPs is realizable and in fact delivers higher performance when compared to a software-based implementation.**

***Keywords*— field-programmable gate array, fast Fourier transform, parallel architectures.**

I. Introduction

In electronic warfare (EW) state of the art, digital receivers are being widely applied for detecting, estimating and classifying incoming radio frequency (RF) signals from radar emitters. An example of wideband digital EW receiver is given by [1], whose only analog components (following the antenna) are the low noise amplifier and the down-conversion mixer. After digitization, the signal is processed by a spectrum analyzer and a parameter encoder uses the frequency information to create a pulse descriptor word (PDW). Consecutively, the PDW is applied in the transmitter identification. An even more audacious approach is also presented by [1], where the down-conversion stage is removed, creating the *Direct RF Sampling* receiver.

Besides upgrading sensibility aspects, processing data as soon as possible and increasing throughput are desirable requirements due to the time-sensitive nature of EW devices. Therefore, the use of field-programmable gate arrays (FPGAs) is an interesting solution that provides high processing capabilities at relative low cost when compared to application specific integrated circuits (ASICs) [2]. The ubiquity of FPGAs in EW and radar research, either for control or processing, is evident. Use cases are shown in [3]–[5] and their references.

Furthermore, analyzing radar signals not only on time domain but also on frequency domain brings significant enhancements in the capabilities of EW receivers, turning easier and more accurate the measurements of pulse widths, instantaneous frequency, amplitude, noise identification and separation of time-coincident incoming pulses [6]. An example of frequency analysis applied to EW is found in [7], where the frequency domain is used to extract changing pulse characteristics and to identify agility of radar signals based on a joint time-frequency representation (TFR), constructed from short time Fourier transforms (STFTs).

Therefore, given the necessity of obtaining the frequency parameters as faster as possible in a digital EW receiver, the objective of this work is to assess the feasibility of an FPGA-based fast Fourier transform (FFT) processor, composed of a bank of parallel STFTs. In this way, the proposed design can be integrated with the spectrum analysis stage of a wideband receiver similar to that suggested by [1].

The problem of reducing the computation time when extracting time-frequency information has been addressed in previous research. A low-complexity and efficient implementation of the STFT is presented in [8], however, it differs from the work presented here for two reasons: first, it requires specific window functions for the STFT; second, the proposed scheme was simulated in software and not implemented on an FPGA.

To develop and verify the FFT parallel processor suggested in the present work, two main components are employed: an FPGA evaluation board and the MATLAB computing environment. The board — in conjunction with its software framework — is used to implement and run the low-level processing in a combination of custom hardware description language (HDL) entities and intellectual properties (IPs), being the latter provided by the FPGA vendor. The MATLAB environment is applied in the input samples generation and output analysis, using TFRs of the data processed by the FPGA.

This paper is organized as follows: section II presents a summarized background on the Fourier transform (FT) and its expansions in the context of a digital computer implementation; section III describes the implementation of the proposed architecture in the FPGA-related environment; section IV discusses the results of the proposed design and its outputs when exercised with test signals; Finally, conclusion remarks and suggestions for future works are presented in section V.

II. Fourier transform and digital signal processing Background

The Fourier transform (FT) is a mathematical process that relates the time-domain description of a signal to an equivalent frequency-domain representation. Specifically, the FT reveals what frequency components are present in the signal and their respective amplitudes and phases [9]. The spectral analysis based on the FT is primarily linked to the stationary signal

C. Barth, canisio@marinha.mil.br; E. Cansio, edgard.cansio@marinha.mil.br. This work was funded by the Navy Research Institute - Brazil, through the project MAGE MK3.

concept. However, the non-stationary nature of real-world signals introduces limitations to this analysis, which requires alternatives for the examination of the time-changing behavior of the signals [10].

A. Short time Fourier transform

One of the techniques used to overcome these limitations is the short time Fourier transform (STFT), where the time signal is divided into small segments of equal duration with the FT is applied separately to each segment. Thus, each local frequency spectrum can be viewed as a function of time and plotted as a spectrogram. This process is mathematically defined in [11] by combining the Fourier integral with a cut-off function known as "window". Due to the consecutive transformations over time, the STFT is also referred as a "sliding window FT".

In the context of the digital signal processing (DSP), the time-frequency analysis must occur on discrete data for both time and frequency variables, which leads to the definition of the discrete Fourier transform (DFT) [12]. Similarly, the STFT function becomes a "sliding window DFT" and its output has two independent discrete quantities for time and frequency. Thus, if $\boldsymbol{x}$ is a real-valued discrete-time signal of length L, and $\boldsymbol{w}$ is a discrete window function of length N, the discrete STFT $\boldsymbol{X}$ of signal $\boldsymbol{x}$ is

$$X(m,k) \triangleq \sum_{n=0}^{N-1} x(n+mH)w(n)e^{-j(2\pi)k\frac{n}{N}} \qquad (1)$$

where:

$H \in \mathbb{N}$ — hop size. This parameter defines the extent in which $\boldsymbol{w}$ is shifted along $\boldsymbol{x}$.

$m \in [0, M]$ — frame index.

$M = \lfloor \frac{L-N}{H} \rfloor$ — maximal frame index. M is also used to contain $\boldsymbol{w}$ inside the time range of $\boldsymbol{x}$.

$k \in [0, N-1]$ — frequency bin. The bin $k = \lfloor \frac{N}{2} \rfloor$ corresponds to the Nyquist frequency.

$n \in [0, N-1]$ — discrete-time index variable.

$\boldsymbol{w}$ — window function. e.g., a rectangular window $w(n) = 1$ for all n.

Therefore, $X(m,k)$ corresponds to the k^{th} complex coefficient of the m^{th} frame, whose unit is the same as the input signal $\boldsymbol{x}$. In other words, each time frame m in (1) represents a spectral vector of size N comprised of the coefficients $X(m,k)$. In conclusion, the computation of a spectral vector is equivalent to a size N DFT and can be efficiently accomplished using the fast Fourier transform (FFT).

B. Fast Fourier transform

Successively to the discrete definition, a class of highly efficient methods for the computation of the FT is derived from the DFT: the fast Fourier transform (FFT) algorithms. The most popular FFT algorithm, attributed to Cooley and Tukey [13], factorizes a size N transform into smaller DFTs, which are then recursively factorized. This process also takes advantage of the symmetry and periodicity of the complex exponential [12] combined with sequential multiplications by a "twiddle factor" [14]. The FFT performance is further enhanced by setting N to a power of 2 or 4 (referred to as *radix-2* and *radix-4*, respectively), which allows for additional simplifications in the computation process [2].

It is worth mentioning that the enhancements of the FFT reduce the complexity of the algorithm to $O(Nlog_2N)$, as opposed to the direct implementation of the DFT, that has an order of $O(N^2)$. Thus, since the STFT requires an FFT of size N for each frame index m, its complexity is $O(MNlog_2N)$ [8]. Apart from this, given that the FFT algorithm consists of successive multipliers and adders, it is particularly well suited for being implemented by FPGA's typical hardware resources, such as DSP slices, look-up tables (LUTs) and block random access memorys (BRAMs).

III. Implementation

This section presents the implementation of the proposed architecture. Firstly, general features and limitations of the components used in this study are discussed. Some important entities developed in VHDL are also explained (i.e., interfaces, a control unit to organize and route samples to the transform bank, and a custom module to instantiate the FFT IPs). Secondly, parameters for transform size, sample size, FFT topology, number of channels and parallel IPs are defined for the tests. The design is synthesized and implemented on the *Zybo* FPGA and resource utilization is evaluated. Finally, procedures for output extraction and post-processing analysis using MATLAB are described.

Although some parameters are chosen at this point of the work, it might also be observed that the architecture is flexible because its components were programmed to be as generic as possible. Consequently, all sizes related to the inputs and their transforms as well as the amount of parallel FFT IPs are scalable. So, the maximum value of a given parameter depends exclusively on the resources offered by the target device.

A. Firmware Architecture Description

The target device used to develop and further evaluate the performance of the FFT processor was the *Digilent Zybo Z7-20 Development Board* [15], that presents a Xilinx Zynq-7000 family [16] as its central element. The Zynq-7000 is a system-on-a-chip (SoC) family that has two main programmable areas: processor system (PS) and programmable logic (PL). In the present work we intend to use only the PL side of the *Zybo*, which consists of an FPGA element based on the Artix-7 XC7Z020 [16]. Thus, the design consists only of a firmware layer and has no embedded software on PS. The development environment used in the project flow (coding, synthesis, programming, debug and analysis) was the *Vivado Design Suite* [17]. Fig. 1 depicts an overview of the firmware architecture implemented on the Zynq-7000 PL.

Considering that the *Zybo* board does not have a high-speed RF front-end and hence is unable to perform analog-to-digital conversion of external signals, the MATLAB software was used to generate discrete-time signals that supply the FPGA hardware. Each sample of the generated signal represents a quantized voltage, which is coded as a 16-bit two's complement number — also known as Q0.15 format [2] — and

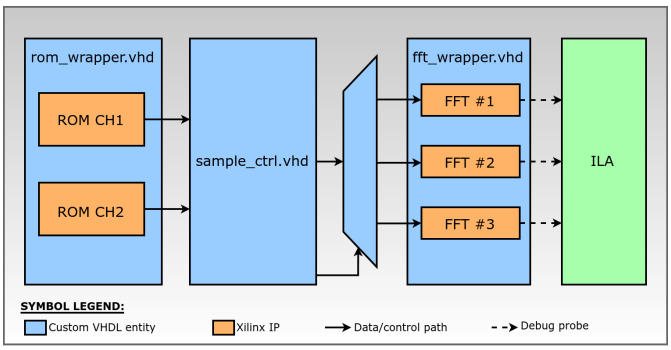


Fig. 1. Firmware architecture.

stored on the *rom_wrapper* entity, comprised of two read-only memories (ROMs) IPs named as "Block Memory Generator (8.4)" in Vivado IP catalog, being one ROM element for each processing channel and one position per sample. Each ROM has 24,576 positions depth by 16-bit width.

The following stage is a full custom entity named as *sample_ctrl*, which performs both ROM data consumption and control of data distribution for *fft_wrapper*. The *sample_ctrl* entity access both ROMs simultaneously with the same sequential addresses until the last memory position, equivalent to the last sample of each channel. Incoming samples are delivered to the demultiplexer data input, whereas its selector input is driven according to an internal counter managed by *sample_ctrl*. The internal counter function is to change samples route, by means of the demultiplexer selector input, at each 8 KSa section, since there is needed three 8 KSa dual-channel FFT to settle the 24 KSa input signals. Fig. 2 depicts the samples distribution.

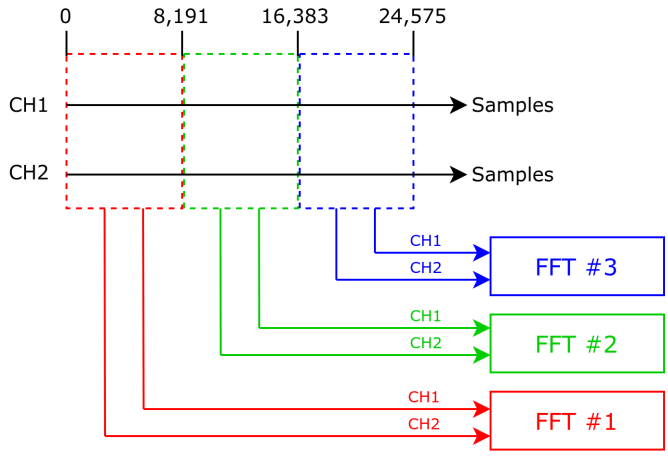


Fig. 2. Samples distribution scheme.

The main hardware entity of the current application is the arrangement of FFTs *fft_wrapper*. This entity is comprised of multiple identical "Fast Fourier Transform (9.1)" [18] IP, available in Vivado IP catalog, which is capable of performing frequency transforms of signals from 8 to 65,536 samples, coming from up to 12 independent channels. Input data is received by the FFT IP through a serial synchronous interface, whose width is designed to support real and imaginary components of all input channels in a multiplexed scheme. As the samples are arriving, an internal buffer is filled up to reach a number of samples per channel equal to the configured transform size. At this moment, when the input buffer is full, the IP starts the transform procedure.

Because an output interface or a memory map to the SoC PS were not implemented, an integrated logic analyzer (ILA) [19] was chosen as the way to monitor and collect data processed by the FPGA, which is then exported to the computer by means of the *Zybo*'s JTAG interface.

B. FPGA Implementation

With the intend to test the architecture, each FFT IP was configured to have two independent input channels with transform size N_{FFT} = 8,192. For the current implementation, this number of points leads to sufficient frequency resolution for input signals whose bandwidths are up to 1 GHz (sampled at some gigahertz, e.g., 2 to 5 GHz). Moreover, these are considered reasonable ranges for down-converted signals in EW systems. Given that it is required to design a fast hardware to perform the FFT, radix-4 was the selected topology for the IP core. Although this implementation requires more logical resources, it presents the lowest latency when compared to other options offered by the design tool. According to Vivado's predictions, there is a trade-off between resources, latency and FFT topology, as can be seen in Table I.

TABLE I
HARDWARE RESOURCES AND LATENCY REQUIRED BY EACH FFT TOPOLOGY FOR N_{FFT} = 8,192 SAMPLES

Topology	DSP Slices	BRAM Tile	Latency (clock cycles)
Radix-4	18	38	30,864
Radix-2	6	34	69,849
Radix-2 Lite	4	34	122,910

Considering that an ILA is essential and that it consumes a significant portion of the PL resources, the maximum count of FFT IPs instantiated by *fft_wrapper* was defined to be three. Furthermore, to demonstrate the parallelism of the conceptual architecture and the functionality of the FFT processor, the input signal should have 24 KSa per independent channel, provided that each IP is able to parallel process a segment of 8 KSa on each of its individual channels.

Table II shows a partial report of the mainly used resources. The values were obtained after the implementation of the entire design, including the ILA. Percentages refer to the total hardware resources available on the *Zybo's* PL.

TABLE II
UTILIZATION REPORT AFTER IMPLEMENTATION STEP

Entity	Slice LUT	Block RAM Tile	DSP Slice
rom_wrapper	93 (0.17%)	23 (16.43%)	0 (0.00%)
sample_ctrl	126 (0.24%)	0 (0.00%)	0 (0.00%)
fft_wrapper	7,564 (14.22%)	57 (40.71%)	54 (24.55%)
ila	1,603 (3.01%)	44 (31.43%)	0 (0.00%)
TOTAL	9,386 (17.64%)	124 (88.57%)	54 (24.55%)

C. MATLAB Data Processing

The output payload of each FFT IP was captured with the aid of the synthesized ILA and its interface through Vivado. Additionally, the ILA front-end was used for debugging, triggering and monitoring of the signals under analysis. Next, the captured datasets were exported as plain text files and imported into MATLAB environment, where a script was

developed to separate the real and imaginary components of each processed sample, as well as organizing the FFT coefficients — $X_i(k)$ — per channel and per IP. Furthermore, to clearly organize the imported data, a 3D matrix was built within the script as depicted in Fig. 3. The algorithm extracts the absolute value of each complex coefficients by consuming pairs of consecutive columns, row by row, until it passes through the whole 3D matrix. The magnitudes are used both for plotting the spectrum of each STFT and for composing the two time-frequency matrices, $X_{CH1}(t, f)$ and $X_{CH2}(t, f)$, one per channel.

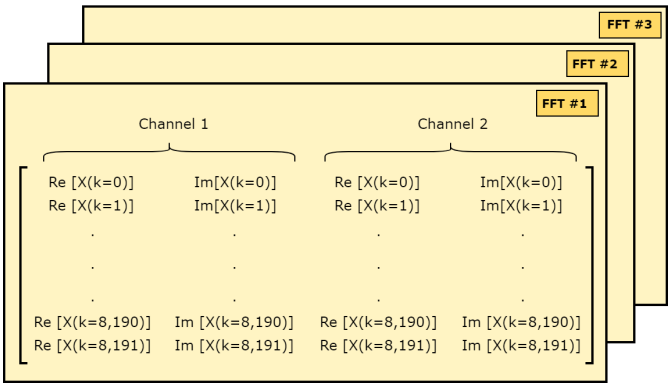


Fig. 3. 3D Matrix arrangement of processed signals.

For plotting the time-frequency representation of the input signals, the FFT magnitudes obtained from each STFT were disposed in matrix format. Whereas the columns represent the three time slices — time resolution of 8,192 columns per time slice of each transform — the rows represent the frequency axis, from 0 to fs, with steps of fs/N — frequency resolution. To maintain consistency with the FFT algorithm and keep the implementation as general as possible, the entire frequency range is considered for the calculations, however, only positive frequencies are shown in this paper, as explained in section IV. Considering a sample frequency of 5 GHz, the frequency resolution obtained is 203.45 KHz. Therefore, the time-frequency matrices look as follows:

$$X(t,f) = \begin{bmatrix} X1(f=0Hz) & \dots & X2(f=0Hz) & \dots & X3(f=0Hz) & \dots \\ X1(f=203.45KHz) & \dots & X2(f=203.45KHz) & \dots & X3(f=203.45KHz) & \dots \\ \vdots & & \vdots & & \vdots & \\ X1(f=5GHz) & \dots & X2(f=5GHz) & \dots & X3(f=5GHz)) & \dots \end{bmatrix} \quad (2)$$

Finally, the time-frequency matrices were processed using the MATLAB function *imagesc*, which interprets each matrix element as a pixel intensity value. Therefore, the output of this function is an image whose the horizontal axis represents time, the vertical axis represents frequency and each pixel intensity represents the transform magnitude at the coordinate in which it is located.

IV. Results

To evaluate the implemented FPGA architecture and the post processing environment, waveforms with known behavior were input to the system. The following subsections discuss the obtained results along with a performance comparison between the FPGA and MATLAB implementations. Regarding the subsequent plots which show frequency output, single-sided magnitude spectra and positive time-frequency representations were adopted, because only real-valued signals were tested.

A. Continuous wave inputs

To first validate the hardware IPs in conjunction with the implemented VHDL and MATLAB codes, a control test with two continuous wave (CW) inputs was executed. The frequencies of the sinusoidal tones were configured at 650 MHz for channel 1 and 1,000 MHz for channel 2. Since the incoming signals do not change frequency neither amplitude along the 24 KSa observation window, all three FFT IPs generated exactly the same frequency tones for each channel, as expected. Fig. 4 depicts the outputs of the CW test.

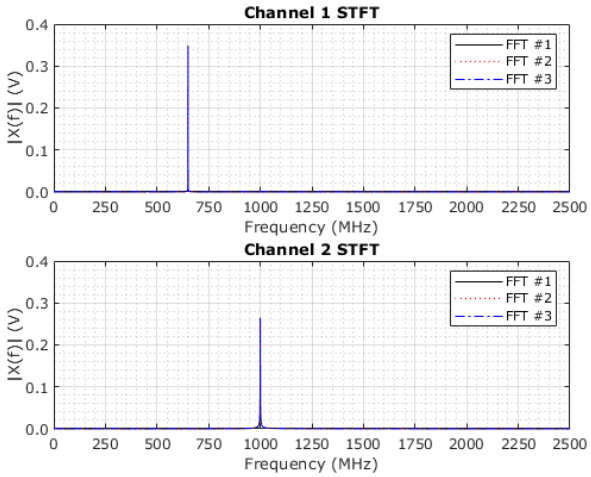
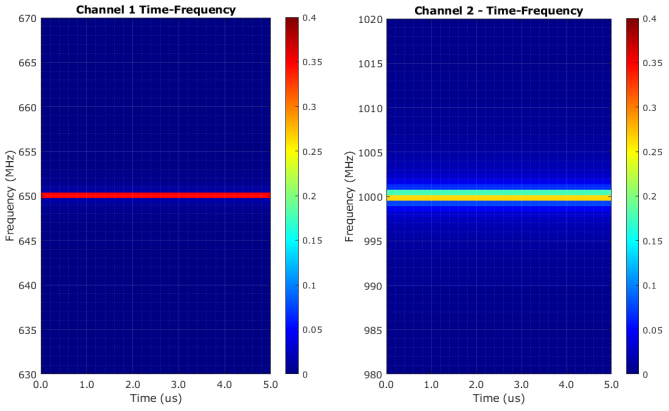


Fig. 4. Output of all three FFT for CW input signals. Top: Channel 1. Bottom: Channel 2

The correctness of the processed FFT matrix is assured by the time-frequency representation of Fig. 5, which shows that the signals on each channel were persistent and continuous along the observation window.



(a) Channel 1 (650 MHz). (b) Channel 2 (1,000 MHz).

Fig. 5. Time-frequency for CW input. Colormap magnitudes in volts.

B. Orthogonal frequency-division multiplexing inputs

To further evaluate the capabilities of the implemented design, a more complex and dynamic input with a multiple carrier signal was injected in the FFT processor. The chosen inputs are similar to an orthogonal frequency-division multiplexing (OFDM) modulation, where the carrier frequencies hop along the time to different bands [20]. For this test, each channel was supplied with three different tones and each tone frequency was modified according to Table III.

TABLE III
FREQUENCY MODIFICATIONS FOR OFDM TEST

Signal	Interval	Carrier Frequency (MHz)
$x_1(n)$	$0 \leq n \leq \frac{N}{3} - 1$	100
		175
		200
	$\frac{N}{3} \leq n \leq 2\frac{N}{3} - 1$	95
		230
		320
	$2\frac{N}{3} \leq n \leq N - 1$	130
		185
		350
$x_2(n)$	$0 \leq n \leq \frac{N}{3} - 1$	200
		260
		460
	$\frac{N}{3} \leq n \leq 2\frac{N}{3} - 1$	400
		640
		700
	$2\frac{N}{3} \leq n \leq N - 1$	190
		350
		370

Fig. 6 depicts the magnitudes of the coefficients processed by each parallel FFT IP on the same spectrum, showing the arrangement of the carriers along the frequency axis. To provide a better visualization, the horizontal axis was limited to 800 MHz (rather than $fs/2$ = 2,500 MHz).

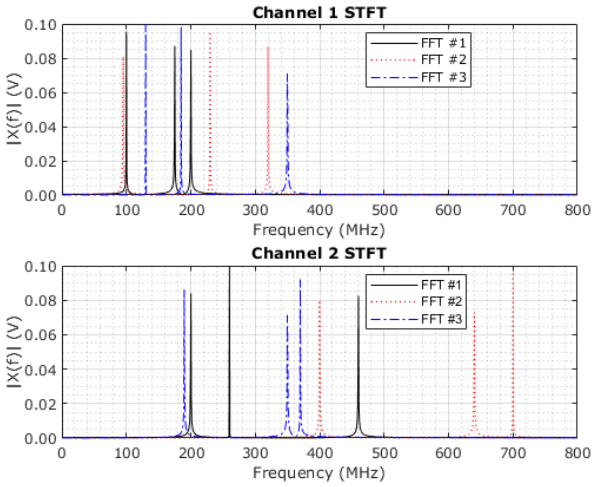
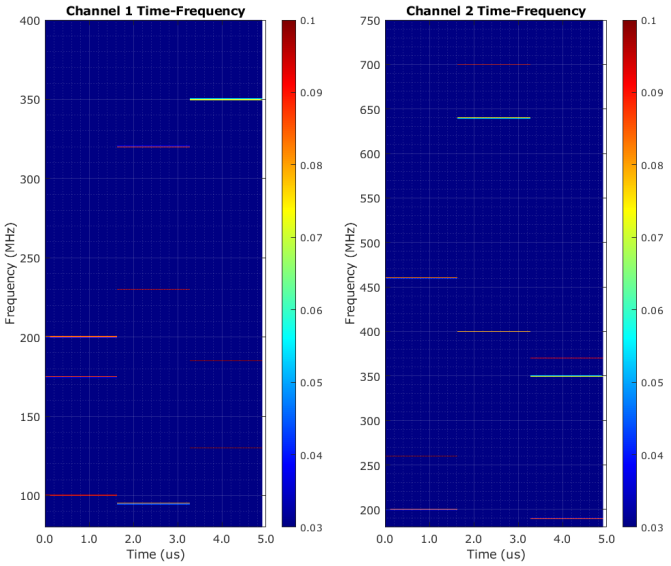


Fig. 6. Output of all three FFT IPs for OFDM. Top: Channel 1. Bottom: Channel 2

Additionally, Fig. 7 illustrates the frequency and magnitude dynamics of each carrier along the time, highlighting the OFDM characteristic of abruptly changing the operating frequencies along the time.

C. FPGA vs. MATLAB: performance comparison

Hardware implementations of digital processing techniques tend to achieve better performance than processor-based solutions by reducing the time required for delivering processed data, since a custom and typically parallel hardware is synthesized to implement the desired function in low-level elements. To verify this property within the context of the FFT processor implementation on the Zynq-7000 PL, there was made a comparison between the elapsed time to transform a 24 KSa signal by the FPGA and by MATLAB, with the latter running on an Intel i7-10700 @ 2.90 GHz processor.



(a) Channel 1. (b) Channel 2.

Fig. 7. Time-frequency for OFDM. Colormap magnitudes in volts.

For discussing the results, it is important to turn clear some architecture differences between these two approaches which have crucial influence on the measured times.

First, on MATLAB side, two 24 KSa discrete-time signals are generated and organized in matrices, which are delivered to three sequential FFT functions per channel, in blocks of 8 KSa, and so are transformed to frequency domain complex coefficients. Second, looking to the FPGA side, the 24 KSa signals of each channel are consumed by two parallels lanes and delivered to three first in, first out (FIFO) buffers — 8 KSa size inside each FFT IP — in such a way that the first FIFO buffer fills when it receives the 8,192nd sample of both channels and the last one fills when it receives the 24,576th sample.

Due to the fact that the FFT IPs are predisposed in a parallel topology, each IP does not need to wait until the others have enough data (i.e., 8 KSa) to start its processing. However, the fact that each IP requires that input data be delivered in a sample-per-sample scheme leads to a behavior that the FFTs processing are not triggered at the same time. Hence, the total elapsed time is greater than the latency of a single IP (ideal case expected for a parallel topology). Fig. 8 shows a time-chart containing each of the three processes performed by FFT IP and their behavior along time, as well as the way that one IP triggers the next one (represented by black arrows).

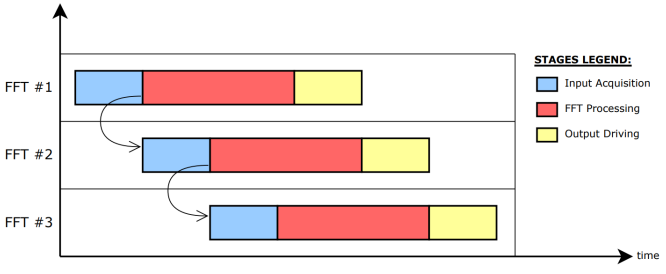


Fig. 8. FFT IP working stages along time.

Considering the differences previously discussed, the FPGA implementation took around of 37.80% of the MATLAB total time to perform the same dual channel FFT (equivalent to 2.46 X faster). Counting only the time effectively used by the FFT IP processor and by the MATLAB sequential FFT functions to transform the same data amount, the performance advantage is even greater, since the FPGA version takes approximately 28.46% of the MATLAB time (3.5 X faster).

V. Conclusion

It is noticed that FPGA devices offer the possibility for fast performing digital signal analysis techniques, specially frequency transforms, with precision and low latency by means of implementing a dedicated and parallel-based hardware for these roles. Naturally, a trade-off situation is raised, since the cost paid for achieving high performance is the hardware resources allocated for the implementations, which is limited to the amount of these resources available on each device. Thus, the objective of presenting a high performance FFT parallel processor for interacting with RF front-ends was achieved, and its scalability depends on the hardware resources offered by the FPGA target device.

About the time-frequency analysis, another trade-off exists when facing time resolution and frequency resolution, due to the fact that for increasing the first one it is needed to increase the number of STFT (e.g., increase the number of IPs), which means reducing the size of each STFT when considering a fixed size signal. Otherwise, smaller STFT leads to reducing the frequency resolution, which in turn is proportional to the number of samples of each STFT. To reach both time and frequency resolution, overlapped windows in time domain can be used, in such a way that the same segment of signal is processed by more that one STFT, according to the overlapping ratio.

For future works based on the proposed architecture and having a SoC as the central element, the PS side can be used as a control element for triggering the FFT processor on the PL side, as well as allowing to more elaborate functionality, such as window overlapping and the use of a cyclical arrangement for transforming larger signals. Furthermore, employing a processor element, such as Zynq PS, offers the advantage of using more complex communications interfaces for trading data from the PL to a high level system, by means of both bare-metal software and embedded operating system functions. Therefore, the ILA does not need to be used as interface to extract data, releasing FPGA resources for growing up the dimensions of the FFT processor. For these reasons, new tests schemes can be closer to real application scenarios.

Based on the resources utilization presented in Table II, it is estimated that two additional FFT IPs can be instantiated by *fft_wrapper* — identical to the three IPs already considered — if the ILA is removed. This modification implies on having an arrangement of five parallel FFTs, which would be able to transform signals of up to 40 KSa at a time or to provide a better time-resolution for incoming signals whose sizes remain the same (i.e., 24 KSa). It is important to note that such scaling in the number of parallel FFTs is only possible due to the fact that *sample_ctrl* entity does not consume the same critical resources of the IPs, such as BRAM and DSP slices, as shown in Table II. Thus, adding two FFT would cost extra 38 BRAMs and 36 DSP slices, leading to a total usage within the FPGA of 84.29% for BRAMs and 40.91% for DSP slices. Finally, due to the parallel processing of the transforms, the three to five FFTs scaling, equivalent to 66.67% of additional IPs, increases the total latency just on 34.68%.

References

[1] P. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, ser. Artech House radar library. Artech House, 2009. [Online]. Available: https://books.google.com.br/books?id=K_T4M-nA6JYC

[2] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, ser. Signals and Communication Technology. Springer, 2007. [Online]. Available: https://books.google.com.br/books?id=7rg0RP0bMgUC

[3] C. Barth, R. A. Romero, and D. J. Fouts, "FPGA implementation of multiple low-rate sampling composite detector," in *2021 18th European Radar Conference (EuRAD)*, 2022, pp. 205–208. [Online]. Available: https://ieeexplore.ieee.org/document/9784556

[4] J. Grajal, O. Yeste-Ojeda, M. Sanchez, M. Garrido, and M. Lopez-Vallejo, "Real time FPGA implementation of an automatic modulation classifier for electronic warfare applications," in *2011 19th European Signal Processing Conference*, 2011, pp. 1514–1518.

[5] C. A. Sessions, R. A. Romero, and D. J. Fouts, "A field-programmable gate array implementation of a cognitive radar target recognition system," in *2022 IEEE Radar Conference (RadarConf22)*, 2022, pp. 1–6.

[6] A. Singh and K. Rao, "Digital receiver-based electronic intelligence system configuration for the detection and identification of intrapulse modulated radar signals," *Defence Science Journal*, vol. 64, no. 2, pp. 152–158, Mar. 2014. [Online]. Available: https://publications.drdo.gov.in/ojs/index.php/dsj/article/view/5091

[7] A. Adam Ahmad, A. Saliu, A. Airoboman, M. Mahmud, and S. Abdullahi, "Identification of radar signals based on time-frequency agility using short-time fourier transform," *Journal of Advances in Science and Engineering*, vol. 1, pp. 1–8, 08 2018.

[8] J.-J. Ding and H. Hu, "Low complexity time-frequency analysis methods for efficient implementation," in *2014 IEEE International Conference on Consumer Electronics - Taiwan*, 2014, pp. 195–196.

[9] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals & Systems (2nd Ed.)*. USA: Prentice-Hall, Inc., 1996.

[10] F. Hlawatsch and F. Auger, *Time-frequency Analysis: Concepts and Methods*, ser. Digital signal and image processing series. ISTE, 2008. [Online]. Available: https://books.google.com.br/books?id=W0DpAAAACAAJ

[11] K. Gröchenig, *Foundations of Time-Frequency Analysis*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2001. [Online]. Available: https://books.google.com.br/books?id=sjN2qq99-WwC

[12] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. Pearson Education, 2011. [Online]. Available: https://books.google.com.br/books?id=EaMuAAAAQBAJ

[13] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: http://www.jstor.org/stable/2003354

[14] W. M. Gentleman and G. Sande, "Fast fourier transforms: For fun and profit," ser. AFIPS '66 (Fall). New York, NY, USA: Association for Computing Machinery, 1966, pp. 563–578. [Online]. Available: https://doi.org/10.1145/1464291.1464352

[15] Digilent Inc., "Zybo Z7 Board Reference Manual." [Online] Available: https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf, 2018, acessed: Jun 30, 2022.

[16] Xilinx Inc., "Zynq-7000 SoC Data Sheet: Overview." [Online] Available: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview, 2018, accessed: Jun 30, 2022.

[17] ——, "Vivado Design Suite User Guide: Getting Started (UG910)." [Online] Available: https://docs.xilinx.com/r/en-US/ug910-vivado-getting-started, 2022, accessed: Jun 30, 2022.

[18] ——, "PG109 Fast Fourier Transform v9.1 LogiCORE IP Product Guide." [Online] Available: https://docs.xilinx.com/r/en-US/pg109-xfft/Fast-Fourier-Transform-v9.1-LogiCORE-IP-Product-Guide, 2022, accessed: Jun 30, 2022.

[19] ——, "Integrated Logic Analyzer v2.0 Data Sheet(DS875." [Online] Available: https://docs.xilinx.com/v/u/en-US/ds875-ila, 2022, accessed: Jul 06, 2022.

[20] S. Haykin and M. Moher, *An Introduction to Analog and Digital Communications*. Wiley, 2007. [Online]. Available: https://books.google.com.br/books?id=OQ1LAQAAIAAJ