P. Paz and M. Garrido, "A 12.8-GS/s 32-Parallel 1 Million-Point FFT," 2024 39th Conference on Design of Circuits and Integrated Systems (DCIS), Catania, Italy, 2024, pp. 1-6, doi: 10.1109/DCIS62603.2024.10769108.

A 12.8-GS/s 32-Parallel 1 Million-Point FFT

Pedro Paz
Department of Electronic Engineering
Universidad Politécnica de Madrid
Madrid, Spain
p.pazm@alumnos.upm.es

Mario Garrido
Department of Electronic Engineering
Universidad Politécnica de Madrid
Madrid, Spain
mario.garrido@upm.es

***Abstract*—This paper presents an FPGA-based 32-parallel 1 million-point fast Fourier transform (FFT) architecture. The proposed implementation achieves a throughput of 12.8 GSps and a latency of 84.9 μs becoming the fastest 1 million-point FFT in literature that fits in a single FPGA. Additionally, the modular design based on computing several sub-FFTs simplifies the architecture and facilitates the design, implementation, debugging, and maintenance of the proposed circuit.**

***Index Terms*—FFT, MDC, FPGA, parallel, pipeline, high throughput, radix-32**

I. Introduction

The fast Fourier transform (FFT) is a signal processing algorithm widely used in various fields such as telecommunications [1] or radio astronomy [2]–[4]. Traditionally, ultra-long FFT implementations, such as 1 million-point FFTs, face significant challenges due to their computational complexity and memory requirements. As the demand for high-resolution signal processing increases, so does the need for FFT architectures that can handle such an amount of data efficiently.

The works in [5], [6] proposed sparse FFT architectures of 1 million points and 0.75 million points, respectively. Sparse FFTs are useful in applications where the energy is concentrated in a few frequencies and the rest is noise. However, they are not suitable for applications where a complete analysis of the spectrum is required. The work in [7] proposed a 128 to 1 million-point reconfigurable FFT architecture on ASIC. This architecture uses external memory and is based on an iterative computation using a reconfigurable radix-2/4/8 butterfly. In [8], several interconnected field-programmable gate arrays (FPGAs) were used to compute the 1 million-point FFT. The first implementation of a 1 million-point FFT architecture in a single FPGA without using external memories was proposed in [9]. This was a serial FFT architecture and achieved a throughput of 233 MSps. However, by searching the literature, no previous work proposed a highly parallel 1 million-point FFT architecture in a single FPGA without the use of external memories.

In this paper, we propose a 32-parallel 1 million-point FFT architecture, which is capable of achieving very high throughput and fits on a single FPGA. Throughout the paper, we explain the design methodology used to address the design complexity of the 1 million-point FFT. We detail the implementation of the high-precision rotators required in the architecture and explain the memory management used to store the rotation coefficients and the data reordering. We also explain the optimization techniques used to increase the throughput of the implementation.

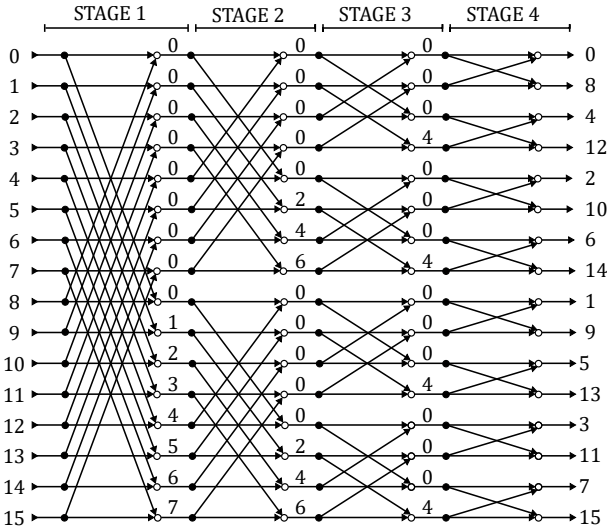

Fig. 1. 16-point FFT flowgraph.

This paper is organized as follows. In Section 2 we review the FFT algorithm, fully parallel FFT architectures, and key FPGA elements such as the DSP slices and memory resources, which are critical in the proposed design. Section 3 describes the proposed architecture. From the methodology used in its design to the implementation details that made it possible to implement the FFT in the FPGA. Section 4 presents the experimental results. Section 5 compares the proposed FFT architecture with previous 1 million-point FFT implementations. Finally, Section 6 presents the conclusions of this paper.

This work was supported in part by Comunidad de Madrid through the call "Ayudas de Estímulo a la Investigación de Jóvenes Doctores de la Universidad Politécnica de Madrid" under Project APOYO-JOVENES-21-TL23SB-116-I4FOMC; and in part by MCIN/AEI/10.13039/501100011033 and "ESF Investing in your future" under Grant RYC2018-025384-I.

II. Background

A. The FFT Algorithm

The N-point discrete Fourier transform (DFT) of an input sequence $x[n]$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1..., N-1, \tag{1}$$

where $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$.

The fast Fourier transform is the name given to the algorithms that allow to compute the DFT reducing its computational cost. The most popular FFT algorithm is the Cooley-Tukey algorithm [10]. This algorithm is based on decomposing the initial DFT into 2-point DFTs. This decomposition adds rotations between each pair of 2-point DFTs.

Fig. 1 represents the flow graph of a 16-point FFT. Each stage computes $N/2$ 2-point DFTs, also called radix-2 butterflies. The numbers between the stages (ϕ) represent the rotations that must be computed between two 2-point DFTs. The rotation angle is computed as $\alpha = -\frac{2\pi}{N}\phi$. The numbers at the input of the flow graph represent the index n of the input sequence, whereas the numbers at the output are the index k of the output frequencies.

B. Fully Parallel FFTs

Several FFT hardware architectures have been proposed over the years [11]. Different architecture types aim to provide different tradeoffs between resource utilization, throughput, and latency.

The fully parallel FFT is a type of FFT architecture obtained as the direct implementation of the flow graphs. This type of architecture is characterized by having a parallelization equal to the number of inputs, i.e., $N = P$, which is the maximum parallelization possible. Thus, these architectures can achieve very high throughput in the range of GSps and low latency.

This type of architecture was explored in [12], where FFT architectures from 8 to 256 points were presented, obtaining the fastest FFT architectures so far. These architectures were implemented using an automatic generator based on FloPoCo [13], which is an open-source generator of arithmetic cores for FPGA.

C. DSP slices

The DSP slices [14], [15] are the modules dedicated to computing arithmetic operations in AMD (Xilinx) FPGAs. The DSP slice can compute operations of the type

$$\mathcal{P} = \pm[(D \pm A) \cdot B \pm C], \tag{2}$$

where $\mathcal{P}$ is the output port of the DSP slice and A, B, C, and D are inputs of different widths. The width of some ports differs depending on the FPGA family.

DSP slices include several pipeline registers which allow them to work at very high clock frequencies.

D. Memories

In AMD (Xilinx) FPGA devices, the main memory elements are called block RAMs [16]. Block RAMs are 36 Kb memories. These memories have two write and two read ports of 36 bits each. Each pair of write and read ports can use an independent clock. The input of the block RAMs can be configured as 1, 2, 4, 9, 18, 36, or 72 bits wide depending on the necessities of the design, adjusting the depth based on the selected port width. Block RAMs can also be initialized.

Since the ultrascale architecture, AMD FPGAs include a new type of memory called UltraRAM. UltraRAMs are 288 Kb memories. UltraRAMs are larger compared to block RAMs, but also less versatile. UltraRAMs have two read and two write ports, but all of them share the same clock. The port width is 72 bits and can not be configured. Thus, the size of the UltraRAMs is 288 Kb = 4 Kb$\times$72. Since UltraRAMs are larger than block RAMs, they require more pipeline stages to work at high clock frequencies, resulting in larger latency for read and write operations. Additionally, UltraRAMs can not be initialized.

III. Proposed 1 Million-Point FFT

A. Building the Architecture

Based on the Cooley-Tukey algorithm [10], an N-point DFT can be decomposed into smaller DFTs. According to it, a 2^{20}-point FFT can be computed as 2 consecutive 1024-point FFTs, with additional rotations between both FFTs. Again, we can apply the Cooley-Tukey algorithm to decompose the partial 1024-point FFTs. This concept is the base of the proposed 1 million-point architecture. Fig. 2 represents the proposed architecture. It is composed of two 1024-point FFTs. Circuits for reordering the data and rotators are required between the both FFTs. In Fig. 2 both 1024-point FFTs are marked with dotted boxes. The permutation circuits are represented as boxes named *Matrix transposition*. The size of the matrix to be transposed is also indicated. The rotators are denoted with ($\otimes$). These rotators use the coefficients stored in ROM memories and the data samples from previous stages of the FFT as input.

Each 1024-point FFT is designed using the same method: it is implemented as two 32-point FFTs with circuits for reordering the data and computing the rotations between the sub-FFTs. The 32-point FFTs are one of the fully parallel FFT cores provided in [12], which processes 32 data in parallel. The proposed 1 million-point architecture uses 4 instances of this core in total. These 32-point FFTs are denoted as R32 in Fig 2.

This design methodology simplifies the design, implementation, and maintenance of extremely large FFTs as the one proposed in this work, since a large part of the code is reused. Note that the 32-point FFTs are used 4 times and both 1024-point FFTs are exactly the same circuit.

The two main challenges of this design are the memory utilization for the implementation of the data reordering circuits and coefficient memories, and to achieve the precision required in the rotations.

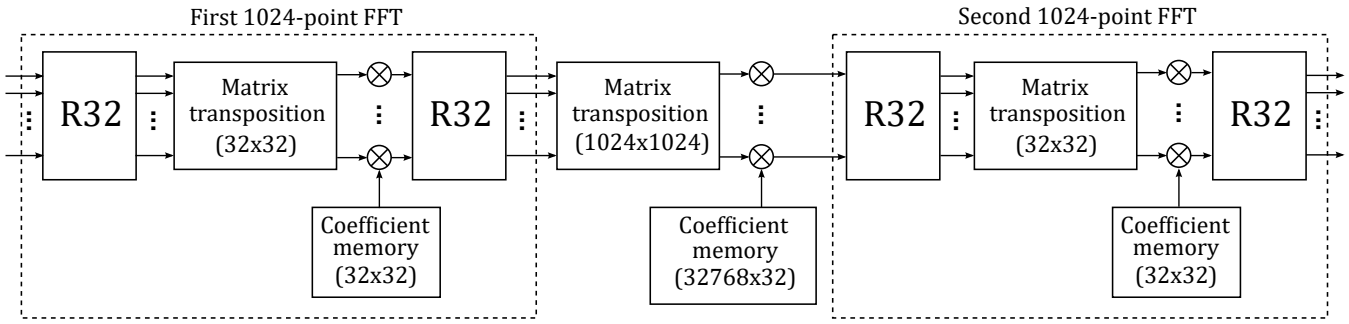

Fig. 2. Proposed 32-parallel 1 million-point FFT architecture.

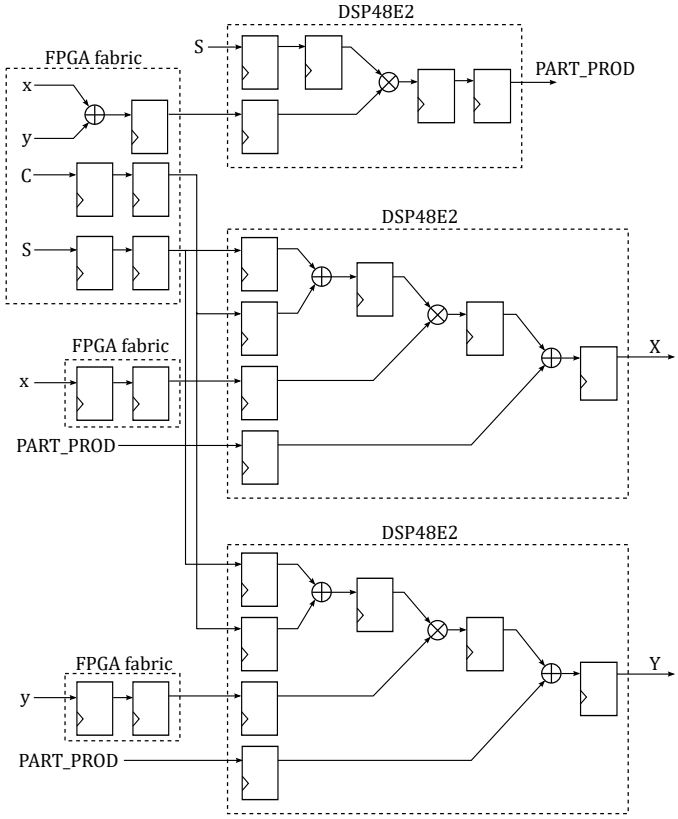

Fig. 3. General rotator for extended coefficient word length.

B. Rotations

Inside each 32-point FFT in the architecture, several rotations are computed. Since the 32-point FFT is a fully parallel implementation, these rotations can be implemented with single-constant multipliers (SCM) [17], [18]. This type of multiplier uses shift-and-add techniques instead of using real multipliers saving FPGA resources.

Inside the 1024-point FFTs, a stage of rotations is required between the two 32-point FFTs. These rotations are not constant, i.e., a rotation by a different angle is computed every clock cycle. Therefore, a general rotator per branch is used instead of single-constant multipliers. The implementation of these complex multipliers is the high-throughput implementation with three DSP slices proposed in [19].

Between the two 1024-point FFTs, another stage of rotations is required. This stage also requires 32 general rotators, one per parallel branch. However, the same implementation used inside the 1024-point FFTs is not suitable. The rotation angles at this stage are obtained by dividing the circumference in 2^{20} parts. The complex multipliers proposed in [19] have a maximum word length of 18 bits for each input. Therefore, to compute the 2^{20}-point FFT with accuracy, we need a higher resolution in these rotations.

Using a CORDIC rotator [20], [21] has been discarded since the truncations computed at the CORDIC stages limit the accuracy achievable by this type of rotator

Fig. 3 presents the rotator designed to compute the rotations between the 1024-point FFTs. This rotator is based on the high-throughput complex rotator in [19]. The DSP slices are marked with dotted boxes and the name of the model, which is DSP48E2. Only the used operators and pipeline registers are drawn in the figure for clarity. The elements implemented using the CLBs of the FPGA are denoted as *FPGA fabric*, and marked with a dotted box. Note that the output of the upper DSP slice is connected to the two lower DSP slices. For clarity, this connection is denoted with the label *PART_PROD*.

The DSP slices in AMD (Xilinx) FPGAs have 25x18 [14] or 27x18 [15] multipliers, depending on the FPGA family. The 18-bit wide port is an input directly to the multiplier, while the larger ports come from the pre-adder. In the high-throughput implementation in [19], the upper DSP uses the pre-adder to compute x minus y and requires the coefficient S to be input at the 18-bit port. We have modified this rotator, computing this subtraction outside the DSP slice, using the logic resources of the FPGA. Then, the result of this subtraction can be input at the 18-bit port, directly to the multiplier, and the coefficient S can be input at the larger port, bypassing the pre-ader. This modification allows to use rotation coefficients of up to 25 or 27 bits, depending on the FPGA family, which takes advantage of the structure of the DSP slices and improves the precision of the rotations without increasing largely the resource utilization.

In this design, we use 20 bits to represent the rotation coefficients for the rotators between the two 1024-point FFTs, as a tradeoff between accuracy and memory utilization.

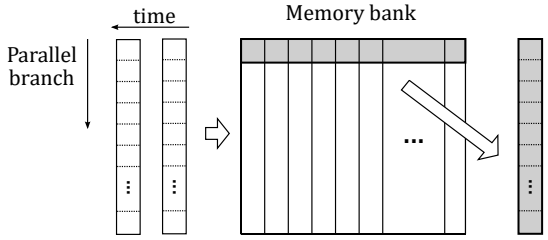

Fig. 4. Example of the operation carried out by the matrix transposition circuits.

C. Data Reordering Circuits

There are two different circuits for data reordering in the proposed architecture. Both compute matrix transpositions [22], [23] but for different matrix sizes. Fig. 4 shows an example of the computation of a matrix transposition. Data samples arrive in parallel as columns to the memory bank. When enough columns are stored to fulfill a complete row, it is read from the memory bank and used as input to the next stages of the FFT.

In the 1024-point FFT, we compute a matrix transposition of 32 by 32. Since the proposed architecture has 32 parallel branches and data is processed in pipeline, samples are stored in memories for 32 clock cycles by columns and then read by rows of the matrix.

Thus, we require enough memory to store 1024 complex samples with a word length of 16 bits for each real and imaginary parts. Therefore, the memory that is required is

$$MT_{32x32} = 32 \cdot 32 \cdot 2 \cdot 16 = 32 \text{ Kb}. \quad (3)$$

This matrix transposition circuit is replicated twice in the architecture, once in each of the two 1024-point FFTs.

Between the two 1024-point FFTs we implement a 1024 by 1024 matrix transposition. Thus, we are required to store $1024 \cdot 1024$ complex samples. Thus, the memory required is computed as

$$MT_{1024x1024} = 1024 \cdot 1024 \cdot 2 \cdot 16 = 32 \text{ Mb}. \quad (4)$$

It can be observed that the 1024 by 1024 matrix transposition requires 1024 times more memory than the 32 by 32 matrix transposition.

D. Memory Utilization

As explained in Section III-C, the matrix transposition circuit between the two 1024-point FFTs requires a large amount of memory. The rotation coefficients also need to be stored in memories. For the rotations between the two 1024-point FFTs, which have a word length of 20 bits for the coefficients, $2^{20} \cdot 2 \cdot 20 = 40$ Mb of memory are required. Adding the memory required for the matrix transposition plus the memory required for storing the coefficients, a total of 72 Mb of memory are required. Block RAMs are usually preferred because they are more versatile and have lower latency. In this design, 3584 block RAMs would be needed for the 1024 by 1024 matrix transposition and the storage of the rotation coefficients of the middle rotator. The target FPGA of this design (XCVU37P) contains a total of 2016 Block RAMs. Therefore, since the device does not have enough block RAMs, we use UltraRAM as well. Since UltraRAMs can not be initialized, they can not be used to store rotation coefficients. Thus, block RAMs are used to store rotation coefficients and the matrix transpositions of 32 by 32, whereas UltraRAMs are used for the 1024 by 1024 matrix transposition.

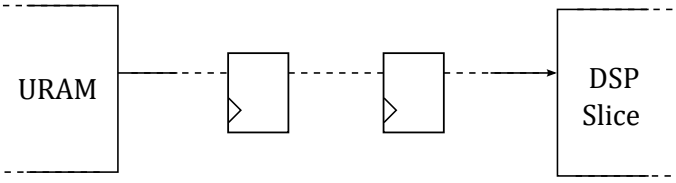

Fig. 5. Additional pipelining stages to reduce the length of interconnection nets.

TABLE I
EXPERIMENTAL RESULTS OF THE PROPOSED 32-PARALLEL 1 MILLION-POINT FFT

Parameter	Value		
N	2^{20}		
P	32		
Data WL	16		
Coef. WL	20		
FPGA	xcvu37p		
f_{CLK} (MHz)	400		
Th (GS/s)	12.8		
FFTs/s	12207		
Latency (clock cycles)	33947		
Latency (μs)	84.9		
P (W)	18.9		
SQNR (dB)	28.4		
Resource	Used	Total	Utilization
BRAM	1286	2016	63.8%
URAM	120	960	12.5%
DSP slices	288	9024	3.2%
LUTs	74859	1303680	5.7%
LUTRAMs	13525	600960	2.25%
FFs	1233360	2607360	4.73%
CLBs	19476	162960	11.9%

E. Implementation Optimizations

In small designs, it is usually assumed that the delay of a circuit is caused mainly by the logic delay. Designers place pipeline registers dividing large combinational circuits into smaller ones to achieve higher clock frequencies and the interconection delays are often depreciated.

In large designs, such as this one, the place and route is a difficult task. Block RAMs, UltraRAMs, and DSP slices are placed in columns inside the FPGA and sometimes, due to the nature of the circuit designed it is not possible to place the elements close enough to each other. This results in large interconnect delays, sometimes larger than the logic delays. This is the case of this design, where a huge amount of memory resources are used. These memory resources are connected, mainly, to a small amount of DSP slices or other circuits, such as counters used to generate the read and write addresses of the memories. To ease the work for the place and route tool, two design techniques have been used.

TABLE II
COMPARISON OF VERY-LONG FFT ARCHITECTURE.

Parameter	[5]	[6]	[7]	[24]	[9]	Proposed
FFT Type	Sparse FFT	Sparse FFT	Memory-based	Pipelined	Pipelined	Pipelined
N	2^{20}	$3^6 \cdot 2^{10}$	2^{20}	2^{16}	2^{20}	2^{20}
P	-	-	2/4/8	32	1	32
Word length (data)	32	12	32	16	16 - 31	16
Word length (coeficients)	32	12	32	16	16 - 31	20
Technology	Virtex 6	ASIC (45nm)	ASIC (40nm)	Virtex 6	Virtex US+	Virtex US+
Clock frequency (MHz)	100	1500	500	143	233	400
Throughput (GSps)	0.86	109.20	0.07	4.58	0.23	12.80
FFTs/s	860	146283	67	69885	222	12207
Latency (cycles)	138646	-	7400000	2098	$\geq 2^{20}$	33947
Latency (us)	1160	-	14800	14.67	4290	84.9
P (W)	-	0.17	-	-	3.44	18.9
SQNR (dB)	-	-	-	-	95.6	28.4

First, critical circuits have been replicated, e.g., the counters used to generate the write and read addresses of the memories have been replicated so that each replicated counter can be placed closer to a group of memories, reducing the net delays.

Second, additional pipeline registers have been added to the critical interconnection nets. Fig. 5 shows how pipeline registers can be placed in the middle of a long net dividing it into several shorter nets, resulting in lower interconnection delays. We have identified two critical paths. First, the nets between the UltraRAMs in the 1024 by 1024 matrix transposition circuit and DSP slices in the adjacent rotators. Second, between the block RAMs containing the rotation coefficients of the rotators between the two 1024-point FFTs and the DSP slices in these rotators.

IV. EXPERIMENTAL RESULTS

Table I shows the experimental results for the proposed 32-parallel 1 million-point FFT architecture. The proposed design has been implemented using VHDL. The experimental results have been obtained using Vivado 2020.2 and the target device is the AMD (Xilinx) Virtex Ultrascale+ XCVU37P-L2FSVH2892E FPGA.

Data samples are represented with 16 + 16 bits using two's complement representation. 20 + 20 bits and two's complement representation are used for the rotation coefficients at the rotators between the two 1024-point FFTs, which are the ones that require more precision. 16 bits are used for other rotation coefficients along the architecture.

The proposed implementation achieves a maximum clock frequency of 400 MHz, thanks to the optimization techniques detailed in Section III-E. This clock frequency results in a throughput of 12.8 GSps and allows to compute 112207 1 million-point FFTs every second.

The latency of an FFT computation is 33947 clock cycles, which results in 84.9 μs at a clock frequency of 400 MHz. In an FFT computation, the first datum can be output once every data is input to the architecture. Thus, the theoretical minimum latency in a 32-parallel 1 million-point FFT is $2^{20}/32 = 32768$ clock cycles. The additional latency in the architecture is due to the deep pipelining, which allows to obtain the clock frequency of 400 MHz.

The dynamic power consumption of the design has been computed with the power analysis tool embedded in Vivado 2020.2 assigning a toggle rate of 50%, obtaining a value of 18.9 W.

The SQNR has been measured according to [25]. First, a random signal is generated. This signal is input both to the VHDL test bench and to a Python script. The Python script computes the FFT using the NumPy library [26], which is used as a reference to compute the SQNR. Then, after the simulation is completed, the SQNR is computed as:

$$SQNR = 10 \log_{10} \frac{\sum \left(|y_{reference}| - |y_{simulation}| \right)^2}{\sum |y_{reference}|^2}, \quad (5)$$

where $y_{reference}$ is the result of the FFT computed in Python and $y_{simulation}$ is the result of the simulation of the proposed architecture. The obtained SQNR is 28.4 dB.

About the resource utilization, it can be observed that the critical resource in the architecture is the memory. More than 60% of block RAMs and 10% of UltraRAMs are required. The utilization of logic and arithmetic resources is much lower in comparison. Between 2% and 6% of utilization for LUTs and Flip Flops and 3.2% for DSP slices are required in the architecture.

V. COMPARISON

Table II compares the proposed FFT architecture with previous 1 million-point FFT architectures and other long FFT architectures. It can be observed that the proposed architecture obtains the highest throughput among FPGA implementations, with 12.8 GSps. Also, only one of the other FPGA implementations obtains a throughput above 1 GSps. This is due to the high parallelization and the deep pipelining implemented in the architecture. The implementation in [6] obtains a higher throughput. However, this implementation uses ASIC technology, which allows to obtain higher clock frequency. Additionally, the implementation in [6] uses a sparse FFT algorithm, instead of the conventional FFT. This algorithm do not compute all the FFT outputs which allows to simplify the computations but can only be used on sparse signals. The proposed implementation also achieves the lowest

latency both in clock cycles and microseconds among 2^{20}-point FFTs, which is 4 times lower than the next lowest latency reported [5]. The implementation proposed in [9] is the only other pipelined 1 million-point FFT implemented in FPGA found in the literature and uses the same FPGA device as the proposed implementation. It can be observed how the increase of the parallelization and the deeper pipelining improves the throughput by more than 55 times and latency by more than 50 times, at the cost of increasing the power consumption. Additionally, [9] increases the word length of the data at each stage, which results in a significantly higher SQNR. When the word length is not increased after each stage, a truncation is computed to avoid overflow. Thus, increasing the word length inside of the FFT architecture allows to obtain higher SQNR values. However, this also increases the resource utilization. Additionally, it affects the modularity of the design, which is one of the key points of the proposed architecture.

VI. Conclusions

In this work, a high-throughput 1 million-point FFT architecture is proposed. The design is characterized by its modularity. This eases the design of such complex circuits and improves their maintenance. We have detailed the design of the rotators and the memory management required to fit in a single FPGA the rotation coefficients and the matrix transposition circuits without using external memories. We also provided the optimization details used to improve the throughout of the architecture.

The resulting architecture achieves the highest throughput and lowest latency among previous FPGA-based 1 million-point FFT implementations in literature while maintaining reasonable resource utilization, power consumption, and SQNR.

References

[1] N. Rajatheva, I. Atzeni, E. Björnson, A. Bourdoux, S. Buzzi, J.-B. Doré, S. Erkucuk, M. Fuentes, K. Guan, Y. Hu, X. Huang, J. Hulkkonen, J. M. Jornet, M. Katz, R. Nilsson, E. Panayirci, K. Rabie, N. Rajapaksha, M. J. Salehi, H. Sarieddeen, S. Shahabuddin, T. Svensson, O. Tervo, A. Tölli, Q. Wu, and W. Xu, "White paper on broadband connectivity in 6G," 6G Research Vision, University of Oulu, Tech. Rep. 10, Jun. 2020.

[2] M. P. Quirk, M. F. Garyantes, H. C. Wilck, and M. J. Grimm, "A wide-band high-resolution spectrum analyzer," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 36, no. 12, pp. 1854–1861, Dec. 1988.

[3] G. Morris and H. Wilck, "JPL 2^{20} channel 300 MHz bandwidth digital spectrum analyzer," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 3, Apr. 1978, pp. 808–811.

[4] C. He, W. Qiang, G. Zhenbin, and W. Hongxing, "A pipelined memory-efficient architecture for ultra-long variable-size FFT processors," in *Proc. IEEE Int. Conf. Comput. Sci. Inf. Tech.*, Aug. 2008, pp. 357–361.

[5] A. Agarwal, H. Hassanieh, O. Abari, E. Hamed, D. Katabi, and Arvind, "High-throughput implementation of a million-point sparse Fourier transform," in *Proc. Int. Conf. Field-Programmable Logic Appl.*, Sep. 2014, pp. 1–6.

[6] O. Abari, E. Hamed, H. Hassanieh, A. Agarwal, D. Katabi, A. P. Chandrakasan, and V. Stojanovic, "27.4 a 0.75-million-point Fourier-transform chip for frequency-sparse signals," in *IEEE Int. Conf. Solid-State Circ.*, 2014, pp. 458–459.

[7] F. Han, L. Li, K. Wang, F. Feng, H. Pan, B. Zhang, G. He, and J. Lin, "An ultra-long FFT architecture implemented in a reconfigurable application specified processor," *IEICE Electron. Express*, vol. 13, no. 13, pp. 1–12, Jun. 2016.

[8] T. Kamazaki, S. K. Okumura, Y. Chikada, T. Okuda, Y. Kurono, S. Iguchi, S. Mitsuishi, Y. Murakami, N. Nishimuta, H. Mita, and R. Sano, "Digital spectro-correlator system for the Atacama compact array of the Atacama large millimeter/submillimeter array," *Publ. Astron. Soc. Japan*, vol. 64, no. 2, p. 29, Apr. 2012.

[9] H. Kanders, T. Mellqvist, M. Garrido, K. Palmkvist, and O. Gustafsson, "A 1 million-point FFT on a single FPGA," *IEEE Trans. Circuits Syst. I*, vol. 66, no. 10, pp. 3863–3873, Oct 2019.

[10] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[11] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst.*, vol. 94, no. 11, pp. 1345–1364, Nov. 2022.

[12] M. Garrido, K. Möller, and M. Kumm, "World's fastest FFT architectures: Breaking the barrier of 100 GS/s," *IEEE Trans. Circuits Syst. I*, vol. 66, no. 4, pp. 1507–1516, Apr. 2019.

[13] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test Comput.*, vol. 28, no. 4, pp. 18–27, 2011.

[14] "Xilinx - 7 Series DSP48E1 Slice," Mar. 2018, https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[15] "Xilinx Ultrascale architecture DSP Slice," Aug. 2021, https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp.

[16] "Xilinx Ultrascale architecture memory resources," Sep. 2021, https://docs.amd.com/v/u/en-US/ug573-ultrascale-memory-resources.

[17] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits Syst. Signal Process.*, vol. 25, no. 4, pp. 225–251, Apr. 2006.

[18] J. Thong and N. Nicolici, "Time-efficient single constant multiplication based on overlapping digit patterns," *IEEE Trans. VLSI Syst.*, vol. 17, no. 9, pp. 1353–1357, Sep. 2009.

[19] P. Paz and M. Garrido, "Efficient implementation of complex multipliers on FPGAs using DSP slices," *J. Signal Process. Syst.*, vol. 95, pp. 543–550, Apr. 2023.

[20] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electr. Comput.*, vol. EC-8, pp. 330–334, Sep. 1959.

[21] M. Garrido, P. Källström, M. Kumm, and O. Gustafsson, "CORDIC II: A new improved CORDIC algorithm," *IEEE Trans. Circuits Syst. II*, vol. 63, no. 2, pp. 186–190, Feb. 2016.

[22] M. Garrido and P. Pirsch, "Continuous-flow matrix transposition using memories," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 9, pp. 3035–3046, Sep. 2020.

[23] B. Zhang, Z. Ma, and W. Luo, "Parallel pipelined architecture and algorithm for matrix transposition using registers," *IEEE Trans. Circuits Syst. II*, vol. 69, no. 3, pp. 1627–1631, 2022.

[24] M. Garrido, M. Acevedo, A. Ehliar, and O. Gustafsson, "Challenging the limits of FFT performance on FPGAs," in *Proc. Int. Symp. Integrated Circuits*, Dec. 2014, pp. 172–175.

[25] D. Guinart, "Deterministic analysis of the accuracy in FFT hardware architectures," Master's Thesis, Dpt. of Electrical Engineering, Linköping University, Jun. 2012.

[26] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.