Memory-Based FFT Architecture with Optimized Number of Multiplexers and Memory Usage

Zeynep Kaya, Mario Garrido, *Senior Member, IEEE* and Jarmo Takala, *Senior Member, IEEE*

***Abstract*—This brief presents a new P-parallel radix-2 memory-based fast Fourier transform (FFT) architecture. The aim of this work is to reduce the number of multiplexers and achieve an efficient memory usage. One advantage of the proposed architecture is that it only needs permutation circuits after the memories, which reduces the multiplexer usage to only one multiplexer per parallel branch. Another advantage is that the architecture calculates the same permutation based on the perfect shuffle at each iteration. Thus, the shuffling circuits do not need to be configured for different iterations. In fact, all the memories require the same read and write addresses, which simplifies the control even further and allows to merge the memories. Along with the hardware efficiency, conflict-free memory access is fulfilled by a circular counter. The FFT has been implemented on a field programmable gate array. Compared to previous approaches, the proposed architecture has the least number of multiplexers and achieves very low area usage.**

***Index Terms*—Memory-based FFT, perfect shuffle, radix-2.**

I. Introduction

THE fast Fourier transform (FFT) is one of the most prominent algorithms in signal processing applications. In some digital systems, the FFT has to be calculated at very high speed. In this case, pipelined FFT architectures [1] are preferred, as they offer continuous flow processing of one or several parallel data. In other systems, instead of speed, the goal is to reduce the area and hardware resources occupied by the architecture. In this case, memory-based FFTs [2]–[15] are good candidates, as they provide a more compact design.

Memory-based FFTs consist of a group of memories and one or several processing elements (PEs) that calculate the butterflies and rotations of the FFT algorithm. The memories and PEs are interconnected by multiplexers, and the FFT algorithm is calculated by loading data from the memories into the PEs and storing the result again in the memories. This process is iterated until the complete FFT is computed. The reuse of the PEs for different stages reduces the number of butterflies and, therefore, the area of the circuit.

Zeynep Kaya is with the Department of Electricity and Energy, Osmaneli Vocational School, Bilecik Seyh Edebali University, 11500 Bilecik, Turkey, e-mail: zeynep.kaya@bilecik.edu.tr.

Mario Garrido is with the Department of Electronic Engineering, ETSI de Telecomunicación, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mail: mario.garrido@upm.es.

Jarmo Takala is with the Faculty of Information Technology and Communication Science, Tampere University, FIN-33014 Tampere, Finland, e-mail: jarmo.takala@tuni.fi.

This work was supported in part by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe" under Project PID2021-126991NA-I00; in part by MCIN/AEI/10.13039/501100011033 and "ESF Investing in your future" under Grant RYC2018-025384-I; and in part by the Scientific and Technological Research Council of Turkey through the International Postdoctoral Fellowship Program under Grant 1059B192200354.

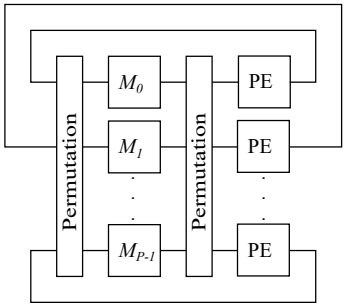

Fig. 1. General structure of a memory-based FFT.

Researchers are trying to improve memory-based FFT architectures by suggesting new memory addressing schemes and decreasing resource requirements such as memory, chip area, etc. [6]–[10]. For an N-point FFT, some approaches use memories of size $2N$ or larger in order to avoid memory conflicts [12]–[15]. However, it is possible to reduce memory requirements even further. This results in approaches with the theoretical minimum memory size of N [5], [6], [16].

Apart from memory, butterflies and rotators, memory-based FFTs include permutation circuits with multiplexers [2]–[7], [12], [14], [16]. These circuits are used for shuffling data between the memory and the PEs. In general, two sets of multiplexers are used as permutation circuits, one before and the other one after the memories [6], [7]. Fig. 1 shows the general structure of these approaches for a P-parallel memory-based FFT, where memories are labeled as M_i. For parallel memory-based FFTs, the number of multiplexers increases significantly with the parallelization, which requires a large amount of hardware resources.

This work presents a memory-based FFT that uses permutation circuits only after memories, which reduces the multiplexer usage significantly. In the proposed architecture, conflict-free memory access is achieved by taking advantage of the perfect shuffle permutation [17]. Using the perfect shuffle has the advantage that the architecture calculates the same permutation at all the iterations of the memory-based FFT. This results in simpler hardware, because the architecture does not need to be reconfigured at each iteration. Furthermore, the control of the architecture is simple, as a simple circular counter is used to generate the memory addresses and control the multiplexers.

Another advantage of the proposed architecture is the fact that the read and write addresses are the same for all the memories in the architecture. This not only simplifies the control, but also allows for merging the memories. Furthermore, the proposed architecture uses the radix-2 decimation-

in-frequency (DIF) FFT algorithm, and the proposed approach is generalized to any number of parallel branches, P, and any power-of-two FFT size, $N = 2^n$.

We have structured the rest of the brief as follows: In Section II, we introduce the permutations in FFT architectures, including the perfect shuffle. In Section III, we present the proposed memory-based FFT architecture. In Section IV, we compare the proposed FFT to the previous state-of-the-art memory-based FFTs. In Section V, we provide implementation details and experimental results. Finally, in Section VI, we summarize the main conclusion of this brief.

II. Background: Bit-Dimension Permutations

There is need to reorder data at each FFT stage and bit-dimension permutations are well suited for this [18]. Bit-dimension permutations define a reorder of $N = 2^n$ data based on a permutation of n bits. The position of each datum is calculated as

$$\mathcal{P} = \sum_{i=0}^{n-1} x_i 2^i, \tag{1}$$

where $x_{n-1}, x_{n-2}, \ldots, x_0$ are dimensions, x_{n-1} being the most significant one and x_0 the least significant one. These dimensions define the data flow. For a data flow of P-parallel data, there exist $p = \log_2 P$ parallel dimensions, which define the parallel branches, and $n - p$ serial dimensions that define data arriving in consecutive clock cycles.

The position in (1) can also be expressed as

$$\mathcal{P} \equiv x_{n-1}, x_{n-2}, \ldots, x_0, \tag{2}$$

where ($\equiv$) is used to relate the decimal and the binary representations of a number. Note the difference between the number of parallel data, P, and the position, $\mathcal{P}$.

In this context, a bit-dimension permutation σ of $u = u_{n-1}, u_{n-2}, \ldots, u_0$ can be defined as

$$\sigma(u) = \sigma(u_{n-1}, u_{n-2}, \ldots, u_0) = u'_{n-1}, u'_{n-2}, \ldots, u'_0 = u', \tag{3}$$

where u' is the permuted form of u according to σ. For example, $\sigma(u_2, u_1, u_0) = u_1, u_2, u_0$ is a bit-dimension permutation of the bits in dimensions x_2 and x_1.

A. Perfect Shuffle, Composition and Inverse

The perfect shuffle [17] is a permutation that calculates a circular rotation of the bits according to

$$\sigma(u_{n-1}u_{n-2}\ldots u_0) = u_{n-2}\ldots u_0 u_{n-1}. \tag{4}$$

When several bit-dimension permutations are calculated in sequence, the resulting permutation is the composition of the permutations. For instance, if $\sigma_a(u_2, u_1, u_0) = u_1, u_2, u_0$ and $\sigma_b(u_2, u_1, u_0) = u_0, u_2, u_1$, then calculating σ_b after σ_a results in $\sigma = \sigma_b(\sigma_a) = \sigma_b \circ \sigma_a$, being

$$\sigma(u_2u_1u_0) = \sigma_b \circ \sigma_a(u_2, u_1, u_0) = u_0, u_1, u_2. \tag{5}$$

If $\sigma(u) = u$, then σ is the identity function, i.e., $\sigma = \mathrm{Id}$.

Finally, the inverse permutation σ^{-1} of a permutation σ is the permutation that fulfills

$$(\sigma^{-1} \circ \sigma)(u) = (\sigma \circ \sigma^{-1})(u) = u. \tag{6}$$

Therefore, if $\sigma(u) = u'$, then $\sigma^{-1}(u') = u$, and $\sigma \circ \sigma^{-1} = \mathrm{Id}$.

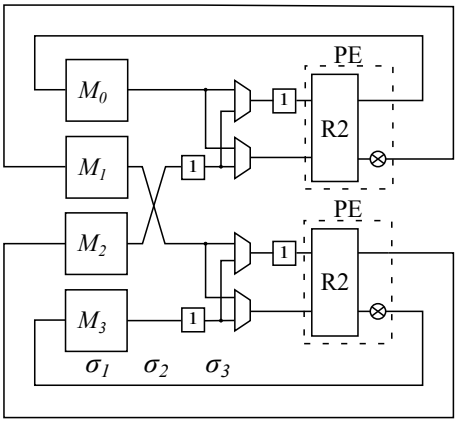

Fig. 2. Proposed basic memory-based FFT architecture for $P = 4$.

III. Proposed Approach

A. Basic Architecture

The basic architecture of the proposed memory-based FFT is shown in Fig. 2 for $P = 4$ data in parallel. It consists of a memory bank with memories M_i that store data and calculate the permutation σ_1, a permutation circuit that calculates σ_2, a permutation circuit that calculates σ_3 and consists of registers and multiplexers, and PEs. Each PE consists of a butterfly (R2), which includes one adder and one subtractor, followed by a complex rotator ($\otimes$).

The memory bank consists of P memories in parallel. This allows for simultaneous read and write operations from all the memories in parallel at each clock cycle. Each memory has N/P addresses, which leads to a total of N memory addresses. These memories not only serve to store the FFT data, but also to permute them according to the permutation σ_1. The calculation of the permutations σ_1, σ_2 and σ_3 leads to $\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1$, which is a perfect shuffle permutation that is calculated at each iteration of the FFT. The permutations σ, σ_1, σ_2 and σ_3 are discussed later in Section III-B.

After the permutation σ, data are processed in the PEs and, then, the results are placed again in the memories. These values are stored in the memory locations that were emptied in the previous reading, which guarantees a conflict-free access.

Once all the iterations of the memory-based FFT are carried out, the results of the last PE calculation are provided as the output of the architecture.

B. Conflict-Free Access

Initially, samples are stored in natural order, i.e., each memory M_q, $q = 0, \ldots, P-1$, stores data with indices $q, P+q, 2\cdot P+q, \ldots, (N/P-1)\cdot P+q$. As a result, the initial position in memories of each index $I \equiv b_{n-1}, b_{n-2}, \ldots, b_0$ is

$$\mathcal{P}_0 \equiv \underbrace{b_{n-1}, b_{n-2}, \ldots, b_p}_{\text{serial (address)}} \,|\, \underbrace{b_{p-1}, \ldots, b_0}_{\text{parallel (memory)}}, \tag{7}$$

where $p = \log_2 P$, bits $b_{n-1}, b_{n-2}, \ldots, b_p$ correspond to addresses, and bits $b_{p-1}, \ldots, b_0$ determine the memories where data are stored.

Butterflies in an FFT architecture operate on pairs of data whose indexes differ in the bit b_{n-s}. Therefore, at each stage

of the FFT, the data position must change. This is achieved with the perfect shuffle permutation

$$\begin{aligned}\sigma(u_{n-1},\dots,u_p|u_{p-1},\dots,u_0) = \\ u_{n-2},\dots,u_{p-1}|u_{p-2},\dots,u_0,u_{n-1}.\end{aligned} \tag{8}$$

Thus, the initial position $\mathcal{P}_0$ is transformed into

$$\mathcal{P}_1 \equiv \underbrace{b_{n-2},b_{n-3},\dots,b_{p-1}}_{\text{serial}}\,|\,\underbrace{b_{p-2},\dots,b_0,b_{n-1}}_{\text{parallel}}, \tag{9}$$

which is the data order at the input of the butterflies at the first stage of the FFT. Note that b_{n-1} is placed in the lowest parallel dimension, which corresponds to the pairs of data that are input to the PEs.

The permutation σ is applied at each iteration of the FFT to provide the correct data into the PE. It is carried out in three steps according to

$$\sigma = \sigma_3 \circ \sigma_2 \circ \sigma_1. \tag{10}$$

The first permutation is

$$\begin{aligned}\sigma_1(u_{n-1},\dots,u_p|u_{p-1},\dots,u_0) = \\ u_{n-2},\dots,u_p,u_{n-1}|u_{p-1},\dots,u_0.\end{aligned} \tag{11}$$

This permutation rotates the serial dimensions according to a perfect shuffle permutation. Note that parallel dimensions do not change, so σ_1 only affects the content of the memories. Thus, we can remove the parallel part from (11) to obtain the following permutation, which is related to the memory addresses:

$$\begin{aligned}\sigma_{mem}(u_{n-p-1},u_{n-p-2},\dots,u_0) = \\ u_{n-p-2},\dots,u_0,u_{n-p-1}.\end{aligned} \tag{12}$$

As this permutation is the same for all the memories, the read and write addresses are also the same for all the memories.

The inverse permutation σ_{mem}^{-1} is the perfect unshuffle

$$\begin{aligned}\sigma_{mem}^{-1}(u_{n-p-1},u_{n-p-2},\dots,u_0) = \\ u_0,u_{n-p-1},u_{n-p-2},\dots,u_1.\end{aligned} \tag{13}$$

It can be proved that $\sigma_{mem} \circ \sigma_{mem}^{-1} = \text{Id}$.

To calculate the permutation σ_{mem} with memories, it must be fulfilled that [19]

$$\sigma_{mem} = \sigma_R^{-1} \circ \sigma_W, \tag{14}$$

where σ_R and σ_W are permutations on the control counter to obtain the read and write addresses, respectively. For the i-th iteration of the FFT, these permutations are obtained as

$$\sigma_{R_i} = \sigma_{W_i} \circ \sigma_{mem}^{-1}, \tag{15}$$

$$\sigma_{W_i} = \sigma_{R_i} \circ \sigma_{mem} = \sigma_{R_{i-1}}. \tag{16}$$

It is worth noting that $\sigma_{W_i} = \sigma_{R_{i-1}}$. This implies that at any iteration data are written in the addresses that are emptied in the previous iteration, which guarantees a conflict-free access to the memory. As input data are written in natural order, then the initial writing address is $\sigma_{W_1} = \text{Id}$, and (14) results in

$$\sigma_{R_1} = \sigma_{mem}^{-1}. \tag{17}$$

The reading and writing addresses are calculated by a permutation of the bits of the circular counter [19], i.e.,

$$R_A = \sigma_{R_A}(c_{n-p-1},\dots,c_0), \tag{18}$$

$$W_A = \sigma_{W_A}(c_{n-p-1},\dots,c_0). \tag{19}$$

As $\sigma_{W_1} = \text{Id}$, the initial write address, W_{A_1}, is chosen to be equal to the the circular counter, i.e., $W_{A_1} = c_{n-p-1},\dots,c_0$. Likewise, the initial write address is calculated from (17) as $R_{A_1} = c_0, c_{n-p-1},\dots,c_1$. Then, W_{A_2} is obtained from (16). As a result, the sequential read and write addresses for n data and P parallel memory are given as

$$\begin{aligned} W_{A_1} = R_{A_{n-p}} = &\quad c_{n-p-1},c_{n-p-2},\dots,c_0, \\ W_{A_2} = R_{A_1} = &\quad c_0,c_{n-p-1},c_{n-p-2},\dots,c_1, \\ W_{A_3} = R_{A_2} = &\quad c_1,c_0,c_{n-p-1},\dots,c_2, \\ &\quad\vdots \\ W_{A_{n-p}} = R_{A_{n-p-1}} = &\quad c_{n-p-2},\dots,c_0,c_{n-p-1}. \end{aligned} \tag{20}$$

The second permutation, σ_2, is calculated after the memories according to

$$\begin{aligned}\sigma_2(u_{n-1},\dots,u_p|u_{p-1},u_{p-2},\dots,u_0) = \\ u_{n-1},\dots,u_p|u_{p-2},\dots,u_0,u_{p-1}.\end{aligned} \tag{21}$$

This permutation rotates the parallel dimensions and is a permutation of the type parallel-parallel (pp) [18]. Thus, it only changes the parallel branches where data flow. For a P-parallel architecture with $p = \log_2 P$ parallel dimensions, data from branch $u_{p-1},u_{p-2},\dots,u_0$ is moved to branch $u_{p-2},\dots,u_0,u_{p-1}$. As this permutation only shuffles the parallel branches, it does not require any hardware.

The third permutation, σ_3, is calculated before the butterflies and corresponds to

$$\begin{aligned}\sigma_3(u_{n-1},\dots,u_p|u_{p-1},\dots,u_0) = \\ u_{n-1},\dots,u_{p+1},u_0|u_{p-1},\dots,u_1,u_p,\end{aligned} \tag{22}$$

which is a a permutation of the type serial-parallel (sp) [18] and rotates both serial and parallel dimensions. This permutation is accomplished with a hardware circuit that consists of P register and P multiplexer.

The calculation of the permutations σ_1, σ_2 and σ_3 creates the perfect shuffle permutation represented by σ.

Fig. 3 presents the data management for an FFT with $N = 16$ points and $P = 4$ parallel data. The data orders at various stages of the design are shown in the upper part of the figure. Also, the figure shows the contents of the memories M_0 - M_3.

Initially, the data are loaded into the memories as natural order in accordance with the writing address W_{A_1} as given in (20). To fulfill the first iteration, these data are read from the memories according to R_{A_1}, which is equal to a circular bit rotation of W_{A_1}. Note that the read and write operations on the memories calculate σ_1 permutation. σ_2 exchanges the two intermediate branches. Finally, σ_3, the sp permutation, provides the necessary order at the input of the butterfly for the second iteration.

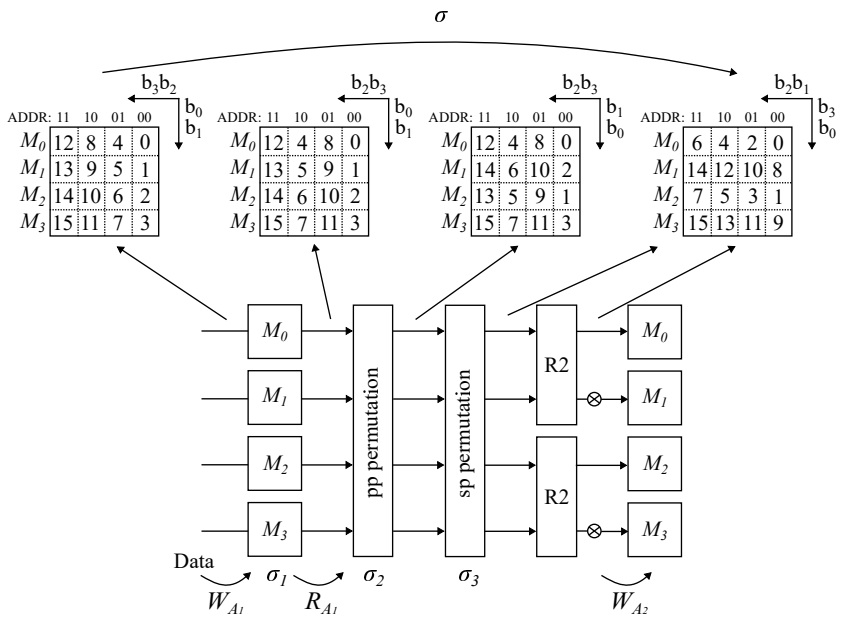

Fig. 3. Data management example for a 16-point memory-based FFT using the proposed approach.

TABLE I
COMPARISON OF MEMORY-BASED FFT ARCHITECTURES.

Approach	Radix	Data Mem. Size	Mem. Banks	Mux.	Complex Mult.	Iterations	Cycles per It.	Processing Time	Data Type
[11]	4	$2N$	$2P$	$9P$	$3P/4$	$(\log_2 N)/2-1$	N/P	$N(\log_2 N)/$ $2P$-N/P	complex
[7]	4	N	P	$60P$	$3P/4$	$(\log_2 N)/2$	N/P	$N(\log_2 N)/2P$	complex
[6]	4	N	P	$4P$	$3P/4$	$(\log_2 N)/2$	N/P	$N(\log_2 N)/2P$	complex
[14]	2/4	$2N$	$2P$	$4P$	$3P/4$	$(\log_2 N)/2$	N/P	$N(\log_2 N)/2P$	complex
[13]	2	$2N+5P/2$	$2P$	$9P$	$P/4$	$\log_2 N$	N/P	$N(\log_2 N-0.5)/P+1$	real
[5]	2	N	P	$4P$	$P/4$	$\log_2 N$	N/P	$N(\log_2 N-1)/P+1$	real
Proposed	2	$N+P$	P	P	$P/2$	$\log_2 N$	N/P	$N(\log_2 N)/P$	complex

IV. COMPARISON

In Table I, we compare the proposed architecture to previous ones as a function of N and P. Radix-4 architectures are placed at the top of the table and radix-2 ones at the bottom.

The table shows that some approaches require a memory with a size in the range of $2N$ addresses [11]–[14], whereas other approaches, including the proposed one, only require memory with size in the range of N addresses. The advantage of the proposed approach is that the read and write addresses are the same for all the memory banks. This allows to merge the banks and, as a consequence, have a very simple control.

The reported numbers of multiplexers are calculated as the equivalent 2-input multiplexers in the permutation circuits. The proposed approach reduces drastically this number with respect to previous approaches, by 75% or more. Likewise, the number of complex multipliers is reduced with respect to previous radix-4 approaches, and is equivalent to that in [5], [13], considering that these works process real-valued inputs.

Finally, for the same N and P, the processing time in radix-4 architectures is approximately half the processing time in radix-2 ones. Even when P is doubled in the proposed architecture to achieve the same processing time as in radix-4 ones, our approach will still have the advantage of a efficient memory usage and a smaller number of multiplexers.

V. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The proposed architecture has been implemented on a Virtex-7 XC7VX330T -1 FFG1157 FPGA. The memories M_0 to M_3 are implemented using block RAMs (BRAMs). Each memory has 1024 addresses of 16+16 bits for the real and imaginary parts, leading to 32Kb per memory. Thus, each memory fits in a 36 Kb BRAM [20] and 4 BRAMs are enough for storing the data. The read and write address of the data memories are generated with a circular counter as in [19].

The sp permutation includes 4 delay registers and 4 multiplexers controlled by the LSB of a simple counter.

After the sp permutation, the PEs calculate the radix-2 butterfly and the rotations. The rotations of the FFT are carried out by two complex multipliers, one for each PE. Each complex multiplier is implemented by using 4 DSP slices, leading to a total of 8 DSP slices in the architecture.

The sine and cosine rotation coefficients are stored in a 64-bit 512-address ROM. In the first stage, each rotator reads different sine and cosine values. Thus, the 64-bit words include two sine and two cosine coefficients, each of 16-bit. For the rest of stages, the coefficients are the same for both rotators.

Table II provides the experimental results of the proposed approach in terms of performance, area, and power consumption, and compares them to previous approaches. The proposed

TABLE II
EXPERIMENTAL RESULTS OF 4096-POINT 4-PARALLEL MEMORY-BASED FFTS ON A VIRTEX-7 FPGA (V7).

Parameter	[5]	[6]⋆	Proposed
N	4096	4096	4096
P	4	4	4
Radix	2	4	2
Iterations	11	6	12
Word length (WL) (bits)	-	24	16
FPGA	V7	V7	V7
Latency (cyc.)	11265	30720	13312
Latency (μs)	27	148	39
Throughput (FFTs/s)	37k	6k	25k
Slices	-	236	210
Slice LUTs	2863	468	465
Slice FFs	2992	585	641
DSP slices	24	26	8
BRAMs	8	11	6
Power (mW)	-	156 @ 208 MHz	208 @ 342 MHz
Area normalized to 16 bits			
Slices @ 16 bits	-	157	210
Slice LUTs @ 16 bits	-	312	465
Slice FFs @ 16 bits	-	390	641
DSP slices @ 16 bits	-	17	8
BRAMs @ 16 bits	-	7	6
Energy per FFT normalized to 16 bits			
Energy/FFT (μJ) @ 16 bits	-	15.4	6.0

⋆: Updated results from [6] for a single 4096-point FFT on a V7 FPGA.
-: Not available.

architecture works at 342 MHz, its latency is 39 μs, and the power consumption is 208 mW. It uses a total of 8 DSP slices, 4 for each complex multipliers, and 6 BRAMs, four of them for data and two of them for rotation coefficients.

Compared to [5], the proposed approach uses significantly less hardware resources at the cost of lower clock frequency and higher latency. To compare to [6], the lower part of the table includes area results normalized to 16 bits and energy per FFT normalized to 16 bits. The proposed approach requires more slices, but less DSPs slices and BRAMs, being the area of both of them similar. However, the proposed approach achieves higher clock frequency, higher throughput, and the energy consumption per FFT is less than half.

VI. CONCLUSIONS

In this brief, we have proposed a radix-2 parallel memory-based FFT architecture based on the perfect shuffle permutation and a novel conflict-free access scheme, which is valid for any FFT size and parallelization. This approach reduces the number of multiplexers and allows for merging the memory banks, which leads to a compact design. Experimental results show that the proposed architecture is hardware-efficient as it achieves high clock frequency, low latency, small area, and low energy per FFT.

VII. ACKNOWLEDGMENT

The authors thank Pedro Paz for providing the implementation of the complex multipliers of the design using DSP slices.

REFERENCES

[1] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst., Survey Papers*, pp. 1–20, Jul. 2021.
[2] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Trans. Signal Process.*, vol. 48, no. 3, pp. 917–921, Mar. 2000.
[3] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 1, pp. 26–30, Jan. 2010.
[4] S.-J. Huang and S.-G. Chen, "A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3c systems," *IEEE Trans. Circuits Syst. I*, vol. 59, no. 8, pp. 1752–1765, Aug. 2012.
[5] Z.-G. Ma, X.-B. Yin, and F. Yu, "A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm," *IEEE Trans. Circuits Syst. II*, vol. 62, no. 9, pp. 876–880, Sep. 2015.
[6] M. Garrido, M. Sánchez, M. López-Vallejo, and J. Grajal, "A 4096-point radix-4 memory-based FFT using DSP slices," *IEEE Trans. VLSI Syst.*, vol. 25, no. 1, pp. 375–379, Jan. 2017.
[7] X. Xiao, E. Oruklu, and J. Saniie, "Fast memory addressing scheme for radix-4 FFT implementation," in *Proc. IEEE Int. Conf. Electro/Inf. Tech.*, Jun. 2009, pp. 437–440.
[8] Q.-J. Xing, Z.-G. Ma, and Y.-K. Xu, "A novel conflict-free parallel memory access scheme for FFT processors," *IEEE Trans. Circuits Syst. II*, vol. 64, no. 11, pp. 1347–1351, Nov. 2017.
[9] H.-F. Luo, Y.-J. Liu, and M.-D. Shieh, "Efficient memory-addressing algorithms for FFT processor design," *IEEE Trans. VLSI Syst.*, vol. 23, no. 10, pp. 2162–2172, Oct. 2014.
[10] Z. Kaya and E. Seke, "A novel addressing algorithm of radix-2 FFT using single-bank dual-port memory," *Circuit World*, vol. 48, no. 1, pp. 64–70, Jan. 2022.
[11] Y. Tian, Y. Hei, Z. Liu, Q. Shen, Z. Di, and T. Chen, "A modified signal flow graph and corresponding conflict-free strategy for memory-based FFT processor design," *IEEE Trans. Circuits Syst. II*, vol. 66, no. 1, pp. 106–110, Jan. 2018.
[12] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Trans. VLSI Syst.*, vol. 19, no. 12, pp. 2290–2302, Dec. 2011.
[13] X.-B. Mao, Z.-G. Ma, F. Yu, and Q.-J. Xing, "A continuous-flow memory-based architecture for real-valued FFT," *IEEE Trans. Circuits Syst. II*, vol. 64, no. 11, pp. 1352–1356, Nov. 2017.
[14] B. Jo and M. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 5, pp. 911–919, May 2005.
[15] J. Chen, J. Hu, S. Lee, and G. E. Sobelman, "Hardware efficient mixed radix-25/16/9 FFT for LTE systems," *IEEE Trans. VLSI Syst.*, vol. 23, no. 2, pp. 221–229, Feb. 2015.
[16] J. Takala, T. Järvinen, and H. Sorokin, "Conflict-free parallel memory access scheme for FFT processors," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 4, May 2003, pp. IV–524–IV–527.
[17] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.
[18] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit-dimension permutations," *IEEE Trans. VLSI Syst.*, vol. 27, no. 5, pp. 1148–1160, May 2019.
[19] 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.
[20] Xilinx, "7 series FPGAs datasheet: Overview," Sep. 2020, https://docs.xilinx.com/v/u/en-US/ds180_7Series_Overview.