A Continuous-Flow Memory-Based Architecture for Real-Valued FFT

Xiu-Bin Mao, Zhen-Guo Ma, Feng Yu, *Member, IEEE*, and Qian-Jian Xing

***Abstract*—This brief proposes a continuous-flow memory-based architecture for fast Fourier transform (FFT) computation for real-valued signals. The proposed architecture is based on a modified radix-2 algorithm, which removes redundant operations to reduce resource usage. A new data-flow graph and address mapping scheme are proposed that satisfy the requirement of continuous-flow operation and minimize the memory usage. The proposed processing element takes advantage of pipelined FFT architectures to avoid bank conflicts in each stage. Compared with prior works, the proposed design has the advantage of supporting continuous-flow operation and normal-order output while minimizing the resource usage.**

***Index Terms*—Real-valued fast Fourier transform (RFFT), memory-based architecture, continuous-flow, normal-order.**

I. Introduction

THE FAST Fourier transform (FFT) is a widely used algorithm in the field of digital signal processing, such as orthogonal frequency-division multiplexing. Most current FFT architectures operate on complex input samples, which are referred to as complex fast Fourier transform (CFFT) architectures. Nowadays, there has been increasing interest in the computation of FFT for real-valued signals (RFFT) because most physical signals are real-valued [1]–[3], such as biomedical signals. The spectrum of the RFFT is Hermitian symmetric, and approximately half of the computations are redundant. When compared to CFFT architectures, a dedicated RFFT architecture requires fewer resources [4], [5].

Current FFT architectures can be divided into two categories: pipelined and memory-based architectures. Pipelined architectures are suitable for high throughput applications but use more hardware resources. In contrast, memory-based architectures require small area and have low power consumption. By employing more processing elements (PEs) in parallel and driving the operating clock frequency to multiples of the system sampling frequency, memory-based architectures can satisfy the demands of continuous-flow and real-time processing [6]. The focus of this brief is on a memory-based RFFT architecture that can support continuous-flow operation and normal-order output.

Manuscript received January 3, 2017; revised February 21, 2017; accepted March 14, 2017. Date of publication March 16, 2017; date of current version November 1, 2017. This brief was recommended by Associate Editor L.-P. Chau.

The authors are with the Department of Instrument Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: maoxiubin@zju.edu.cn; 850501@zju.edu.cn; osfengyu@zju.edu.cn; xingqianjian@zju.edu.cn).

Digital Object Identifier 10.1109/TCSII.2017.2683642

Many memory-based CFFT architectures that support continuous-flow operation have been presented in [6]–[8]. In these architectures, well designed addressing schemes were proposed to avoid bank conflicts and to limit the memory size to $2N$ words. The traditional memory-based RFFT architectures are based on the packing algorithm, which calculates an N-point RFFT using the architecture of an $N/2$-point CFFT [2]. However, this kind of architectures require additional reordering and post-possessing stages to obtain the final results. An efficient algorithm to calculate RFFT has been proposed in [4], which is called the modified radix-2 algorithm. This algorithm removes redundant operations and divides the complex data stream into real and imaginary parts. Thus, only half of the output frequencies are computed, and the word length of the required memory can be reduced by a factor of 2 compared with CFFT architectures.

Recently, some memory-based RFFT architectures have been proposed that are based on the modified radix-2 algorithm [5], [9]. A conflict-free memory access scheme was proposed in [5] to achieve a low area-time complexity, and an architecture that included a new stage partition scheme was proposed in [9] to reduce the number of computational cycles. However, both proposed architectures result in an irregular output address mapping, which can not support continuous-flow operation. This brief proposes a new data-flow graph that supports continuous-flow operation for RFFT computations based on the modified radix-2 algorithm. In addition, a PE is proposed that takes advantage of pipelined FFT architectures to avoid bank conflicts and maximize resource utilization. Compared with prior works, the proposed design supports continuous-flow operation and normal-order output while maintaining a resource efficient architecture.

The rest of this brief is organized as follows. Section II reviews the radix-2 RFFT algorithm and presents a new data-flow graph to support continuous-flow operation. The proposed architecture is presented in Section III. Finally, Section IV compares the proposed design with prior works.

II. Continuous-Flow Operation

The FFT of a N-point sequence of $x(n)$ is defined as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \ 0 \le k \le N-1, \tag{1}$$

where ${W_N}^{nk}$ is the twiddle factor. The RFFT considers $x(n)$ to be a real sequence, then the output $X(k)$ is conjugate symmetric:

$$X(k) = X^*(N-k). \tag{2}$$

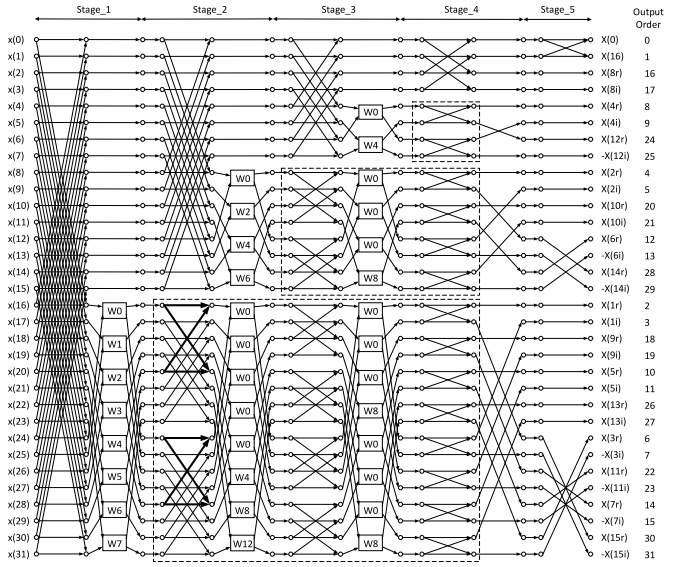

Fig. 1. Proposed data-flow graph of a 32-point RFFT.

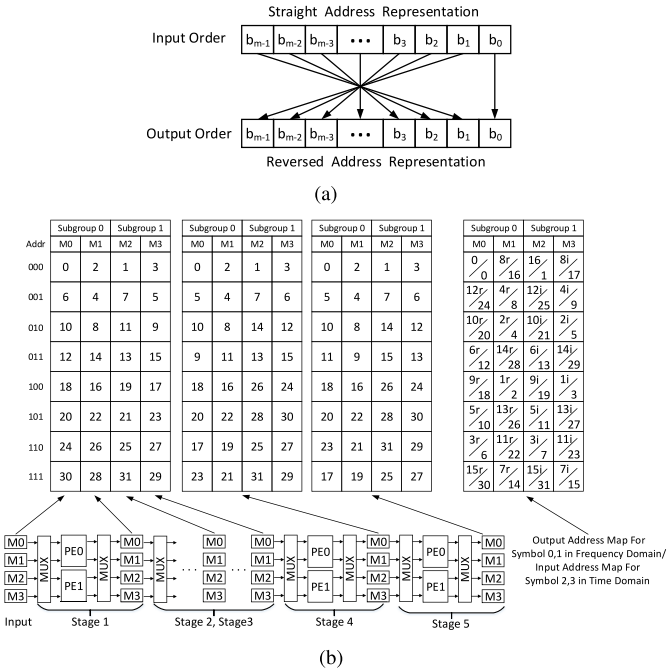

Fig. 2. (a) Address reversal map for a 2^m-point RFFT. (b) Data management example for a 32-point RFFT with two radix-2 PEs.

In a typical memory-based continuous-flow FFT architecture, three types of storage are required, namely input buffering, storage of the intermediate results, and output reordering [6]. Hence, storage for $3N$ words are required. However, the input and output buffers can be merged by a well-designed addressing scheme, and the memory requirements can be reduced to $2N$ words [6], [10]. In these addressing schemes, the input sample of the next symbol is stored at the location from which the output sample of the current symbol has just been read, which is referred to as a concurrent I/O operation.

An inherent problem in the modified radix-2 RFFT data-flow graph is the scrambled output order, which is different from that of a bit-reversal one [4]. The scrambled output order makes it difficult to support concurrent I/O to merge the input and output memory. Moreover, it requires additional operations to obtain a normal-order output.

A new data-flow graph for RFFT computation is proposed to support concurrent I/O operation, as shown in Fig. 1. The data-flow graph is based on the stage partition scheme in [9]. For output frequencies greater than $N/2$, Eq. (2) has been used to get the conjugated result. Shuffling permutations are performed in the last two stages to re-sort the output frequencies into bit reverse order. The output order is shown in the last column of Fig. 1, which results in a normal-order output in the frequency domain. For an $N(=2^m)$-point RFFT, let $\{b_{m-1}, b_{m-2}, \ldots b_1, b_0\}$ represent the binary indexes of the operators in each stages. Then, the address reversal map for the proposed flow graph is shown in Fig. 2(a).

The flow graph is regular and can be generalized to other RFFT sizes that are a power-of-two. The shuffling permutations in the second last stage are performed by exchanging the sample at location $L = \{b_{m-1}, b_{m-2}, \ldots b_l, b_{l-1}, b_{l-2}, \ldots b_1, b_0\}$ with the sample at location $L = \{b_{m-1}, b_{m-2}, \ldots b_l, b_0, b_{l-2}, \ldots b_1, b_{l-1}\}$, where $l = \lfloor log_2(L) \rfloor$. The shuffling permutations in the last stage are performed by exchanging the sample at location $\{b_{m-1}, b_{m-2}, \ldots b_l, 1, b_{l-2}, \ldots b_1, b_0\}$ with the sample at location $\{b_{m-1}, b_{m-2}, \ldots b_l, 1, \overline{b}_{l-2}, \ldots \overline{b}_1, b_0\}$, where $\overline{b}_x$ denotes the negation of b_x.

To calculate an $N(=2^m)$-point RFFT with $P(=2^p)$ parallel radix-2 butterflies, the $2N$-word memory is divided into two groups and operates in a ping-pong mode. Each group of memory is partitioned into $2P$ memory banks. Here, we further map the even-indexed and odd-indexed data into different sub-memory groups, and each subgroup contains P memory banks. A bank and address mapping scheme derived from [11] is applied:

$$Subgroup = b_0, \tag{3}$$

$$Bank = \left\{b_{m-1}, \ldots b_{qp+2}, b_{qp+1}\right\} \oplus \cdots \oplus \left\{b_{2p}, \ldots b_{p+2}, b_{p+1}\right\} \oplus \left\{b_p, \ldots b_2, b_1\right\}, \tag{4}$$

$$Address = \left\{b_{m-1}, b_{m-2}, \ldots b_{p+1}\right\}, \tag{5}$$

where $q = \lfloor (m-2)/p \rfloor$ and $\oplus$ represents modulo-p addition. The physical bank index is formed by combining the subgroup index and bank index. According to Fig. 2(a), this creates a symmetric digit-reverse address map in each subgroup.

An example of the data management similar to that in [12] is shown in Fig. 2(b). The first three stages of the RFFT are processed using in-place strategy, and the permutations in the last two stages lead to data reordering in the memory according to Fig. 1. The writing addresses are the delay copies of the previous reading addresses in each stage, since there are certain delays in the processing element. This means that the intermediate results are written back to the addresses, in which the contents have already been read in previous cycle. The address generation relates to Eqs. (3)-(5) is detailed in Section III-B. Consecutive symbols that access the same memory group alternate between the straight and reversed address representations, e.g., symbol 0 and symbol 2. It can be seen that concurrent I/O is performed

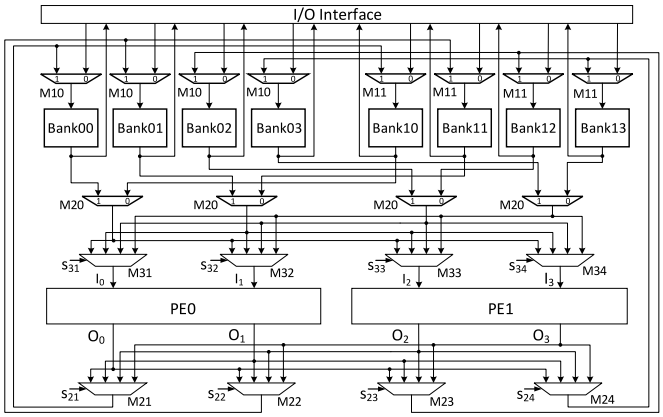

Fig. 3. Block diagram of the proposed RFFT processor.

between symbols to merge the input and output buffers. Hence, continuous-flow operation can be achieved with a $2N$-word memory.

III. Proposed Architecture

The high-level architecture of the proposed RFFT processor with two PEs is shown in Fig. 3. The address generation unit and the twiddle factor ROM have been omitted for clarity. The following sections detail how the processor operates.

A. Processing Element

As shown in Fig. 1, the dashed squares indicate the complex data flow graph, which divides the data into real and imaginary parts. The two bolded butterflies correspond to the real part and imaginary part of one complex butterfly, and they are referred to as the related butterflies. Typically, the related butterflies are computed simultaneously because the outputs of the two butterflies should enter the same complex multiplier.

As we can see from the largest dashed square, the index distance between the upper/lower input of the related butterflies is 2^{m-2} in each stage, while the index distance between the upper and lower inputs of each butterfly is 2^{m-s-1}, where s represents the index of the stage. The four input operators of the related butterflies have binary indexes that differ in b_{m-2} and b_{m-s-1}, where b_{m-s-1} varies as the stage changes. To avoid bank conflicts between the related butterflies, the bank and address mapping should also vary as the stage changes. The PE proposed in [9] simultaneously computes the related butterflies and shuffles the output locations with a form that corresponds to the stage and a counter. The shuffling leads to varying bank and address mappings at each stage to avoid bank conflicts. However, the shuffling also leads to a scrambled output order, which is incompatible with the concurrent I/O scheme required to support continuous-flow operation.

To avoid bank conflicts and maintain the address mapping scheme proposed in Section II, the related butterflies are scheduled to consecutive cycles. This is achieved with a PE that takes advantage of the pipelined FFT architecture. As shown in Fig. 4(a), the PE consists of one butterfly, 3-word delay elements, and a half complex multiplier (HCM). The HCM contains two real multipliers, one real adder, two multiplexers, and 2-word delay elements, as shown in Fig. 4(b).

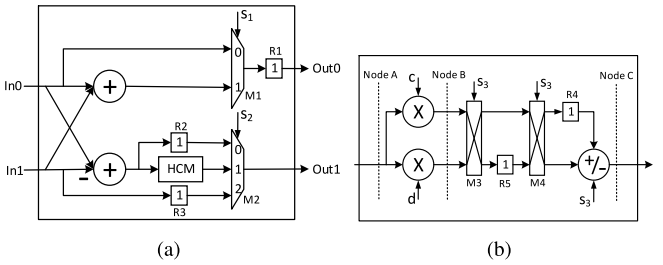

Fig. 4. (a) Block diagram of the proposed processing element. (b) Block diagram of the HCM.

TABLE I
Data Sequence of HCM

cycle	Node A	Node B	R4	R5	Node C	s_3
1	x_{1r}	$c \cdot x_{1r}$, $d \cdot x_{1r}$	-	-	-	0
2	x_{1i}	$c \cdot x_{1i}$, $d \cdot x_{1i}$	$c \cdot x_{1r}$	$d \cdot x_{1r}$	$c \cdot x_{1r}$-$d \cdot x_{1i}$	1
3	x_{2r}	$c \cdot x_{2r}$, $d \cdot x_{2r}$	$d \cdot x_{1r}$	$c \cdot x_{1i}$	$c \cdot x_{1i}$+$d \cdot x_{1r}$	0
4	x_{2i}	$c \cdot x_{2i}$, $d \cdot x_{2i}$	$c \cdot x_{2r}$	$d \cdot x_{2r}$	$c \cdot x_{2r}$-$d \cdot x_{2i}$	1
5	x_{3r}	$c \cdot x_{3r}$, $d \cdot x_{3r}$	$d \cdot x_{2r}$	$c \cdot x_{2i}$	$c \cdot x_{2i}$+$d \cdot x_{2r}$	0

By controlling the multiplexers ($M3$ and $M4$), the HCM can process one complex multiplication in two consecutive cycles. A similar implementation of the HCM can be found in [13], and the difference lies that the design in [13] places the delay elements and multiplexer before the multipliers and adder. The data sequence of the HCM is shown in Table I. The multipliers and adder in the HCM are shared in the time-multiplexed approach. Thus, the related butterflies and the following complex multiplication can be processed in two consecutive cycles by one PE. In each cycle, the PE works on a single butterfly and half of the complex multiplication.

Noting that, the index difference between the upper and lower inputs of each butterfly is a power of two, therefore, the binary indexes of the two input operators only differ by one bit. For each single butterfly, Eqs. (3) and (4) will map its upper and lower inputs to different physical banks, which have a bank index or subgroup index that differ by one bit. In this way, bank conflicts are avoided in one single butterfly. Hence, by scheduling the related butterflies to consecutive cycles, we avoid bank conflicts under the proposed bank and address mapping scheme that supports continuous-flow operation.

B. Address Generation

This section proposes a feasible address generation unit for the proposed RFFT processor. In each stage, butterflies are processed in a top-down order based on the data-flow graph with certain constraints. In the first $(m-2)$ stages, the related butterflies are computed in consecutive cycles. In the $(m-1)$th stage, the two butterflies that exchange their output locations are computed simultaneously. In the last stage, the operators that exchange the locations are read from and written to the corresponding memory locations in consecutive cycles.

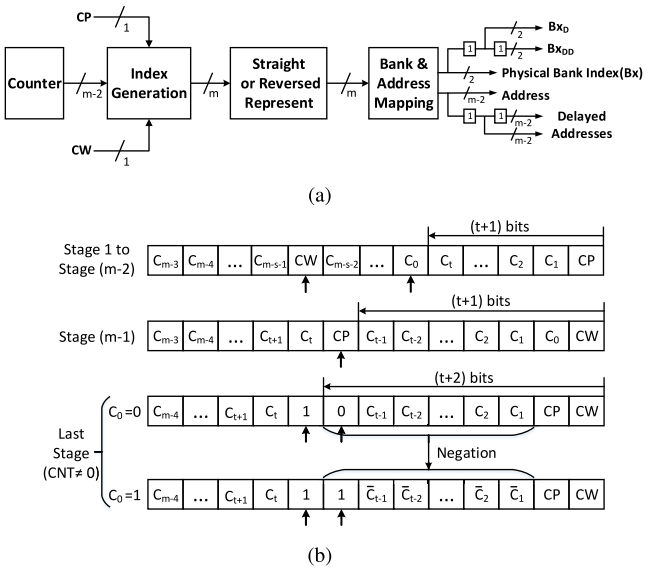

Fig. 5. (a) Block diagram of the address generation unit. (b) The index generation scheme.

TABLE II
CONTROL OF THE MULTIPLEXERS IN PES

Stage	Input cycle	PE0			PE1		
		s_1	s_{2*}[1]	s_3	s_1	s_{2*}[1]	s_3
1 to (m-2)	any	1	1	C_0	1	1	C_0
m-1	any	1	0	-	1	0	-
m	0	1	0	-	0	2	-
	else	0	2	-	0	2	-

[1] Control signal s_2 is a one-cycle delay copy of s_{2*}.

TABLE III
CONTROL OF THE OUTPUT MULTIPLEXERS

Stage	Output cycle[1]	s_{21}, s_{22}, s_{23}, s_{24}
1 to (m-2)	any	$B0_D$, $B1_D$, $B2_D$, $B3_D$
m-1	0	$B0_D$, $B1_D$, $B2_D$, $B3_D$
	else	$B0_D$, $B2_D$, $B1_D$, $B3_D$
m	0	$B0_D$, $B1_D$, $B2_D$, $B3_D$
	1	$B2_D$, $B3_D$, $B0_D$, $B1_D$
	even	$B2$, $B3$, $B0$, $B1$
	odd	$B2_{DD}$, $B3_{DD}$, $B0_{DD}$, $B1_{DD}$

[1] The output cycle is one cycle later than the input cycle in Table II.

The address generation unit is designed hierarchically, as shown in Fig. 5(a). For the proposed processor with two PEs, first, the m-bit index of each operator is generated according to a $(m-2)$-bit counter, i.e., $CNT = \sum_{i=0}^{m-3} C_i 2^i$, and two one-bit counters, i.e., *CP* and *CW*, where *CP* is the counter for the two PEs and *CW* is the counter for the input ports of each PE. The four operators that are processed simultaneously have indexes that only differ in *CP* and *CW*. The generated indexes should use either the straight or reversed representation, depending on the mode of operation. Finally, physical bank indexes and addresses are generated from the indexes based on Eqs. (3)-(5).

The index generation unit is the key module of the address generation unit. We define a basic index as $IDX = \{CNT, CP, CW\}$. In stage s, the indexes are generated by inserting certain bits of *IDX* into other bit locations according to a control signal t, which is defined as:

$$t = \begin{cases} m-s-2, & for \quad CNT < 2^{m-s-1} \\ \lfloor log_2(CNT) \rfloor, & for \quad CNT \geq 2^{m-s-1} \end{cases} \tag{6}$$

where $\lfloor log_2(CNT) \rfloor$ can be implemented by a leading-zero circuit. The index generation scheme is shown in Fig. 5(b). The arrows in the figure represent a bit insertion. In the first $(m-2)$ stages, the indexes are generated by inserting *CW* before C_{m-s-2}, and then inserting C_0 into a bit location according to $(t+1)$. In the $(m-1)$th stage, *CP* is inserted into a bit location according to $(t+1)$. In the last stage, if $CNT = 0$, the index is *IDX*. Otherwise, when *CNT* is even, two binary bits are inserted according to $(t+2)$, and when *CNT* is odd, an additional t bits starting from C_1 are negated to form the index. The last stage takes $N/8$ cycles, because only half of the samples need computation or storage location exchanging.

C. Control of Multiplexers

Table II shows the control of the multiplexers inside the PEs. The control signals s_1, s_2 are used to select the path and decide whether the butterfly or HCM is bypassed in each PE. In the first $(m-2)$ stages, the butterflies and the HCMs are not bypassed and signal s_3 is used to control the HCM in each PE. In the $(m-1)$th stage, no multiplication is performed and the HCMs are bypassed. In the last stage, all butterflies and HCMs are bypassed after the first cycle.

The two memory groups in Fig. 3 work in concurrent I/O mode and computation mode in a ping-pong manner. The multiplexers denoted $M10/M11$ select the I/O interface when the following memory group is working in concurrent I/O mode, and select the intermediate results when the following memory group is working in computation mode. The multiplexers denoted $M20$ select the memory group which is working in the computation mode. The control of these multiplexers alternate for consecutive symbols.

The inputs of the PEs are denoted as I_0 to I_3 in Fig. 3. We define $B0$ as the physical bank index of I_0, and $B0_D$ is a one-cycle delay copy of $B0$ and $B0_{DD}$ is a two-cycle delay copy of $B0$. A similar definition is applied to $B1$, $B1_D$, etc. The delayed physical bank indexes are shown in Fig. 5(a).

The 4-input multiplexer groups before and after the PEs in Fig. 3 are denoted as the input and output multiplexers, and have the same architecture. The input multiplexers are controlled using the physical bank indexes of I_0, I_1, I_2, I_3, i.e., $B0$, $B1$, $B2$, $B3$, throughout the whole process. If the same control is applied to both the input and output multiplexers, no permutation is performed, and this is referred to as an in-place operation. This correspond to the first $(m-2)$ stages in Fig. 1 and the control of the output multiplexers in the first $(m-2)$ stages are shown in Table III.

The permutations in the last two stages of the flow graph are realized by exchanging the control signals of the output multiplexers, e.g., exchanging the control signal s_{22} and s_{23} will exchange the storage locations of O_1 and O_2 in the second last stage. A similar operation is applied to the last stage as shown in Table III, except that the control signals after the second cycle are formed by alternately using the current bank indexes and the delayed bank indexes.

TABLE IV
COMPARISON OF DIFFERENT WORKS

	Proposed(2 PEs)		[9]		[5]		[6]		[6]		[10]	
Radix	radix-2		radix-2		radix-2		radix-2		radix-4		radix-$2/4$	
Parallel Process	4		4		4		2		4		4	
Complex Adders	2		2		2		2		8		8	
Complex Multipliers	1		1		1		1		3		3	
Memory	(2N+10)*W[1]		2N*W		2N*W		2N*2W		2N*2W		2N*2W	
Continuous Flow	Yes		No		No		Yes		Yes		Yes	
N	Cycle	Transistors	Cycle	Transistors	Cycle	Transistors	Cycle	Transistors	Cycle	Transistors	Cycle	Transistors
256	481	69754	448	68794	512	68794	1024	117946	256	159250	256	159250
1024	2433	217210	2305	216250	2560	216250	5120	412858	1280	454162	1280	454162
4096	11777	807034	11265	806074	12288	806074	24576	1592506	6144	1633810	6144	1633810

[1] Parameter W represents the word length.

IV. COMPARISON

In Table IV, we compare the proposed design with prior works. Some CFFT architectures that support continuous-flow operation have been proposed in [6] and [10]. When these architectures are applied to RFFT computations, they are found to contain redundant operations and require significantly more resources. The dedicated RFFT architecture proposed in [5] reduces the memory requirement by a factor of two compared to the CFFT architectures. To minimize the number of computational cycles, the RFFT architecture proposed in [9] uses a new stage partition and reduces the number of computational cycles by almost one stage. The proposed architectures in [5] and [9] all use a memory of $2N$ words. However, neither of the designs supports concurrent I/O operation and they require an additional N-word memory as the output buffer to support continuous-flow operation. The scrambled output order in these designs also requires additional processing to obtain a normal-order output. Our proposed design with two PEs uses the same number of complex adders and complex multipliers as that in [9]; moreover, the proposed design supports continuous-flow operation and normal-order output. We consider each delay element in the proposed PE in Fig. 4 to take one memory word and the two PEs in the proposed design lead to a 10-word memory overhead, but this is tolerable, especially when compared to an additional N-word memory. If all the multiplexers are realized by 2-input multiplexers, the total number of 2-input multiplexers of the proposed design is 50(36 for permutation while 14 in PEs). For an N-point RFFT, the number of computational cycles in the proposed design is $N/4 * (log_2 N - 0.5) + 1$. In a continuous-flow architecture, the computation time of the RFFT should be no more than the sampling time, and according to the discussion in [6], by driving the operating clock frequency to treble of the sampling frequency, the continuous-flow operation can be achieved for a RFFT size equal to or less than 4096 with the proposed processor.

We also compare the proposed design with prior works in the transistor level. We choose the word length to be 16 bits, and according to [14], the 16-bit SRAM, 16-bit multiplier, and 16-bit adder require 96, 4153, and 505 transistors, respectively. Complex multipliers are assumed to be implemented by four real multipliers and two real adders. As shown in Table IV, when the size of the RFFT increases, the memory usage dominates the number of transistors. Considering that the design in [5] and [9] requires an additional N-word memory to support continuous-flow operation, the proposed design achieves continuous-flow operation and normal-order output while requiring fewer transistors.

REFERENCES

[1] M. Ayinala and K. K. Parhi, "FFT architectures for real-valued signals based on radix-2^3 and radix-2^4 algorithms," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 9, pp. 2422–2430, Sep. 2013.

[2] H.-F. Chi and Z.-H. Lai, "A cost-effective memory-based real-valued FFT and Hermitian symmetric IFFT processor for DMT-based wire-line transmission systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 6. Kobe, Japan, May 2005, pp. 6006–6009.

[3] M. Ayinala and K. K. Parhi, "Parallel-pipelined radix-2^2 FFT architecture for real valued signals," in *Proc. Conf. Rec. 44th Asilomar Conf. Signals Syst. Comput.*, Pacific Grove, CA, USA, Nov. 2010, pp. 1274–1278.

[4] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.

[5] M. Ayinala, Y. Lao, and K. K. Parhi, "An in-place FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 10, pp. 652–656, Oct. 2013.

[6] 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. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 12, pp. 2290–2302, Dec. 2011.

[7] R. Radhouane, P. Liu, and C. Modlin, "Minimizing the memory requirement for continuous flow FFT implementation: Continuous flow mixed mode FFT (CFMM-FFT)," in *Proc. IEEE Int. Symp. Circuits Syst. Emerg. Technol. 21st Century*, vol. 1. Geneva, Switzerland, 2000, pp. 116–119.

[8] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 1, pp. 26–30, Jan. 2010.

[9] 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, Exp. Briefs*, vol. 62, no. 9, pp. 876–880, Sep. 2015.

[10] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 5, pp. 911–919, May 2005.

[11] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 5, pp. 312–316, May 1992.

[12] M. Garrido, M. Á. Sanchez, M. L. Lopez-Vallejo, and J. Grajal, "A 4096-point radix-4 memory-based FFT using DSP slices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 375–379, Jan. 2017.

[13] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson, "The serial commutator FFT," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 10, pp. 974–978, Oct. 2016.

[14] J. Liu, Q. Xing, X. Yin, X. Mao, and F. Yu, "Pipelined architecture for a radix-2 fast Walsh–Hadamar–Fourier transform algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 11, pp. 1083–1087, Nov. 2015.