A Parallel Radix-2^k FFT Processor Using Single-Port Merged-Bank Memory

Jian Wang, Xianbin Li, Guangteng Fan and Zhouhui Tuo
National Innovation Institute of Defense Technology
Academy of Military Science, Beijing
Email: wangjian710108@126.com, lixianbincn@163.com, fanguangteng@163.com, 13974874321@163.com

***Abstract*—This paper presents an area-efficient radix-2^k FFT processor employing single-port memory, where the deployed memory is merged into 4 banks for arbitrary 2^k-parallel computation. The proposed design enables the FFT input/output to operate in the parallelism equal to that of internal processing, and it paves the way for gaining high-throughput capability. Moreover, the in-place data caching strategy is available to allow the overlap between caching input data and supplying FFT results, which can further enhance throughput without consuming additional area. Theoretical and experimental comparisons demonstrate the proposed FFT processor can surpass the published related work in throughput while preserving high area efficiency.**

I. Introduction

Fast Fourier transform (FFT) is a fundamental tool in digital signal processing. Over the past decades, designers have actively investigated the hardware realization of FFT and presented two solutions: pipelined architectures and memory-based (MB) architectures. In general, the pipelined design is popular in throughput-intensive applications [1] while MB FFT design is favored in area-constrained scenarios. Driven by the ever-increasing throughput demand, state-of-the-art MB FFT modules have been focused on providing an enhanced throughput of one or multiple clock rates [2]–[7], which offer an alternative tradeoff between area and throughput.

The throughput of a MB FFT processor can be improved by increasing the computing parallelism P_c. A large P_c triggers high-rate data interaction between the processing element (PE) and memory, making the conflict-free data access schedule indispensable. This issue was considered in [2] for radix-2^k processors with dual-port (DP) RAMs, and then generalized to the mixed-radix case [8]. To achieve further area reduction, recently single-port (SP) RAMs become an efficient substitute for DP ones in MB FFT modules [5]–[7], [9], while it complicates conflict-free data access due to the asynchronous reading and writing operations on a RAM. Prior work has been done to tackle this problem [6], [9]. Nevertheless, for a pipelined high-speed PE, the conflict-free solution in [6] involves numerous separate RAM instances that degrade the area efficiency due to the increasing number of peripheral circuits. Although the merged-bank memory structure is utilized in [9] to curtail the number of memory banks, it does not adjust to the high-parallel input/output (I/O) data and thus imposes restrictions on throughput. Furthermore, the in-place data caching is not supported by these FFT processors using SP RAMs, which allows the overlap between receiving FFT input and supplying in-order FFT output without occupying additional memory.

The foregoing factors motivate the work in this paper. We present a radix-2^k MB FFT processor with the computing parallelism $P_c = 2^k$ and the following advantages: (i) The I/O parallelism of FFT can reach P_c, thus its restriction on throughput is eliminated; (ii) The SP RAMs are deployed and merged into 4 banks for an arbitrary P_c and I/O parallelism, which preserves the high area efficiency; (iii) The in-place data storage is available to upgrade the throughput capability.

The paper is organized as follows. Section II introduces the top-level design of the proposed FFT processor. In Section III, the detailed schedule for conflict-free data access is elaborated, which is followed by theoretical and experimental comparisons in Section IV. Finally, Section V concludes the work.

II. Top-Level Design

The radix-2^k FFT algorithm computes an N-point ($N = 2^n$) FFT through $M = \lceil n/k \rceil$ stages, where $\lceil\cdot\rceil$ is ceiling operator. The radix order k_m, $m = 1, \cdots, M$ of each stage is

$$k_1 = n - k(M-1) \le k,\ k_2 = \cdots = k_M = k. \qquad (1)$$

We additionally let $k_0 = 0$. The FFT input, the operands and computing results of each stage, as well as the FFT output can be numbered using the data index $d \in \{0, 1, \cdots, N-1\}$. At stage m, the operands constitute $N/2^{k_m}$ radix-2^{k_m} butterflies, and the data indices related to the $t+1$th butterfly constitute the vector

$$\mathbf{b}_{m,t} = [\mathcal{I}_m(t), \mathcal{I}_m(t)+\varepsilon_m, \cdots, \mathcal{I}_m(t)+(2^{k_m}-1)\varepsilon_m]^{\mathrm{T}} \quad (2)$$

with $\varepsilon_m = N/2^{\sum_{i=0}^{m} k_m}$, $t = 0, 1, \cdots, N/2^{k_m} - 1$. The array $\mathcal{I}_m$ is defined as

$$\mathcal{I}_m = \bigcup_{v=0}^{2^{\sum_{u=0}^{m-1} k_u}-1} \{v\varepsilon_{m-1} : 1 : v\varepsilon_{m-1} + \varepsilon_m\}, \qquad (3)$$

where $\{u : \Delta : v\}$ represents the Δ-incremental integer sequence in the range $[u, v)$.

Corresponding author: Xianbin Li, lixianbincn@163.com.
This work was supported by National Natural Science Foundation of China (NSFC) under Grant 61801503.

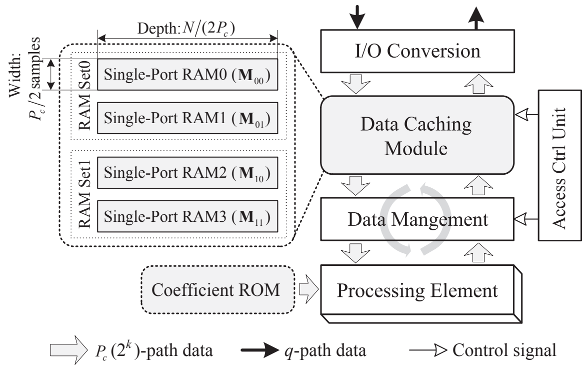

Fig. 1. Top-level architecture of the proposed FFT processor.

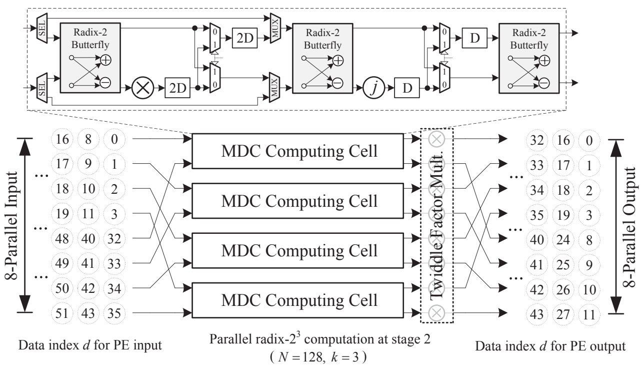
Radix-2 Butterfly
SEL
MUX
2D
D
8-Parallel Input
MDC Computing Cell
Twiddle Factor Mult.
8-Parallel Output
Data index d for PE input
Parallel radix-2^3 computation at stage 2 ($N=128$, $k=3$)
Data index d for PE output

Fig. 3. The PE structure for both radix-2^2 and radix-2^3 computation.

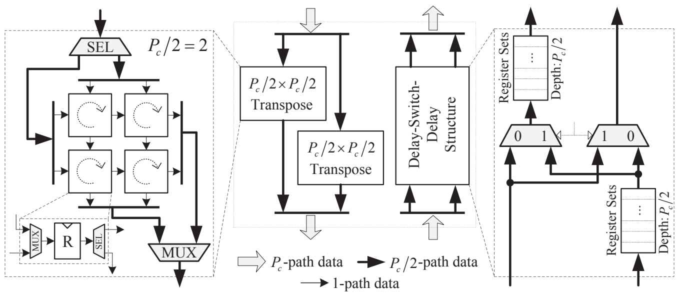

Fig. 2. Hardware structure of the data management unit (DMU).

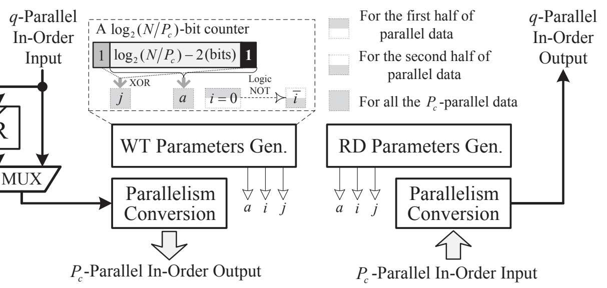
q-Parallel In-Order Input
A $\log_2(N/P_c)$-bit counter
$\log_2(N/P_c)-2$(bits)
XOR
Logic NOT
For the first half of parallel data
For the second half of parallel data
For all the P_c-parallel data
q-Parallel In-Order Output
R
MUX
WT Parameters Gen.
RD Parameters Gen.
Parallelism Conversion
P_c-Parallel In-Order Output
P_c-Parallel In-Order Input

Fig. 4. Hardware structure of the I/O conversion unit.

Example: For radix-2^3 FFT with $N=128$, the radix orders of three stages are $k_1=1$ and $k_2=k_3=3$. At stage 2, $\varepsilon_2=8$ and $\mathcal{I}_2=\{0,1,2,3,4,5,6,7\}\cup\{64,65,66,67,68,69,70,71\}$. Accordingly, the 3rd radix-2^3 butterfly at stage 2 has $\mathbf{b}_{2,2}=[2,10,18,26,34,42,50,58]^{\mathrm{T}}$.

The designed FFT module handles $P_c/2=2^k-1$ butterflies in parallel each time. For stages 1 to $M-1$, the butterflies specified by $\mathbf{b}_{m,t}$, $\mathbf{b}_{m,t+1}$, $\cdots$, $\mathbf{b}_{m,t+P_c/2-1}$ are concurrently processed, where t successively takes the element of the array

$$T_m=\bigcup_{v=0}^{2^{\sum_{u=0}^{m-1}k_u}-1}\mathrm{strd}_2\big(\{v\varepsilon_m:\tfrac{P_c}{2}:(v+1)\varepsilon_m\}\big),\qquad(4)$$

$\mathrm{strd}_2(\cdot)$ here denotes the stride-by-2 permutation. In terms of stage M, the butterflies associated with $\mathbf{b}_{M,t}$, $\mathbf{b}_{M,t+\mathrm{rev}_{n-k}(1)}$, $\cdots$, $\mathbf{b}_{M,t+\mathrm{rev}_{n-k}(P_c/2-1)}$ are processed in parallel, where $\mathrm{rev}_w(u)$ is the w-bit reversal of u, and t traverses the array

$$T_M=\mathrm{strd}_2(R_0\cup R_{2^{n-2k}+1})\cup\mathrm{strd}_2(R_{2^{n-2k}}\cup R_1)\qquad(5)$$

with $R_x=\{\mathrm{rev}_{n-2k}(u)+x|\,u=0,1,\cdots,N/2^k-1\}$ under $N>2^{2k}$. Particularly, $T_M=\{0,1\}$ if $N=2^{2k}$, and $T_M=\{0\}$ if $N=2^{2k-1}$. The minimum FFT length for the proposed design is $N_{\min}=P_c/2\cdot 2^k=2^{2k-1}$, since no less than $P_c/2$ radix-2^k butterflies need to be included.

The top-level design of the proposed radix-2^k FFT processor is shown in Fig. 1. It has an I/O parallelism of q with $q\le P_c$. The access control unit executes an elaborate schedule to gain conflict-free data reading and writing. The data management unit (DMU) in Fig. 2 is used to regulate the data order. PE employs $P_c/2$ radix-2^k multipath delay commutator (MDC) cells handle $P_c/2$ butterflies and gain the computing parallelism $P_c=2^k$. In case of $k_1<k$, by letting the data flow through the last k_1 components of each MDC cell, PE can be used for radix-2^{k_1} computation as well, as illustrated in Fig. 3. The latency of PE set to 2^{k_m} clock cycles for radix-2^{k_m} computation, and this can be attained by suitably pipelining the operation. The I/O data and intermediate results are stored in an N-word data caching module (DCM), which is composed of 4 SP RAMs. The RAM bank is denoted as $\mathbf{M}_{ij}$, $i,j=0,1$, which has a $N/(2P_c)$ depth and a $P_c/2$-sample data width.

III. Conflict-Free Data Access Schedule

To allow conflict-free data access, a mapping rule that converts the data index $d\in\{0,1,\cdots,N-1\}$ into RAM identifers $i,j\in\{0,1\}$ and physical address $a\in\{0,1,\cdots,N/(2P_c)-1\}$ should meet the following constraints:

C1: No more than two RAM identifer pairs $[i,j]$ and no more than two physical addresses a can be involved in data reading or writing.

C2: For the concurrent data reading and data writing, they should employ different RAM identifer pairs $[i,j]$.

C3: Within the computation of a stage, a physical address for data writing cannot be prior to its use for data reading.

The constraint **C1** aims at eliminating conflicts within data reading or data writing operations. On this basis, **C2** and **C3** are verified to avoid collisions between reading and writing.

A. Scheduling of Input Data

The q-parallel input samples are regulated to the P_c-parallel form by the parallelism conversion unit in Fig. 4, and the latter are written to DCM using the mapping rule

$$i = \left\lfloor \frac{2d}{P_c} \right\rfloor \bmod 2,\; j = \left(\left\lfloor \frac{2d}{N} \right\rfloor + \left\lfloor \frac{d}{P_c} \right\rfloor \right) \bmod 2, \\ a = \left\lfloor \frac{d}{P_c} \right\rfloor \bmod \frac{N}{2P_c}. \tag{6}$$

Based on (6), j and a for the concurrent writing data are identical, implying **C1** is satisfied. In the I/O conversion unit illustrated in Fig. 4, i, j, a formulated above can be produced through simple logic operations on the digits of a $\log_2(N/P_c)$-bit counter, which is driven by the enable signal of the P_c-parallel input samples.

B. Data Access at Stage 1 to $M-1$

By following the butterfly processing order given in Section II, the mapping rule in (6) enables the data reading at stage 1 to satisfy **C1**, as the P_c-parallel operands for PE computation will be mapped to the same i, a. The data reading at stage m ($2 \le m \le M-1$) exploits a new mapping rule as below:

$$i = \left\lfloor \frac{2d}{P_c} \right\rfloor \bmod 2,\; j = \left(\left\lfloor \frac{2d}{\varepsilon_{m-1}} \right\rfloor + \left\lfloor \frac{d}{\varepsilon_{m-1}} \right\rfloor \right) \bmod 2 \\ a = \left\lfloor \frac{d}{P_c} \right\rfloor \bmod \frac{\varepsilon_{m-1}}{2P_c} + \left\lfloor \frac{d}{\varepsilon_{m-1}} \right\rfloor \cdot \frac{\varepsilon_{m-1}}{2P_c}. \tag{7}$$

(7) ensures the salification of **C1** for data reading, as parallel operands for PE also have the same i, a at stages 2 to $M-1$.

To cache the PE output of stages 1 to $M-2$, (7) become suitable if replacing ε_{m-1} with ε_m. In this case, the concurrent writing samples utilize the same i, and exact two diverse a are involved each time, implying the data writing satisfies **C1**. To further meet **C2**, **C3**, the delay between the continuous data reading and data writing should be confined to 2^{k_m} clock cycles, which is exactly the PE computing latency.

For stage $M-1$, the PE output first flows through the delay-switch-delay (DSD) component in DMU, as shown in the right part of Fig. 2, after which the reordered samples are stored using the mapping rule:

$$i = \left\lfloor \frac{d}{P_c} \right\rfloor \bmod 2,\; j = \left\lfloor \frac{2d}{P_c} \right\rfloor \bmod 2,\; a = \left\lfloor \frac{d}{2P_c} \right\rfloor. \tag{8}$$

The parallel data will have the same i, a from (8), making **C1** satisfied for data writing. Taking account of the $2^{k_{M-1}-1}$-clock-cycle latency of the DSD component, the delay between continuous data reading and data writing arrives at $3 \cdot 2^{k_{M-1}}$ clock cycles, which allows the satisfaction of **C2** and **C3**.

i, j, a formulated by the above mapping rules can be attained based on a $\log_2(N/P_c)$-bit counter, which is driven by the enable signal of data reading or data writing. By partitioning and swapping the digits of the counter, data access parameters are generated with logic operations, as shown in Fig. 5.

C. Data Access at Stage M

The data reading of stage M is executed based on (8). To meet **C1** during data reading while supplying PE operands as the desired order, DMU performs $P_c/2 \times P_c/2$ block transposition on the upper- and lower-$P_c/2$ data streams, as depicted in the left part of Fig. 2, whose latency is 2^{k-1} clock cycles. In terms of caching the PE output, the first $N/2$ samples are directly sent to DCM, while the remaining ones

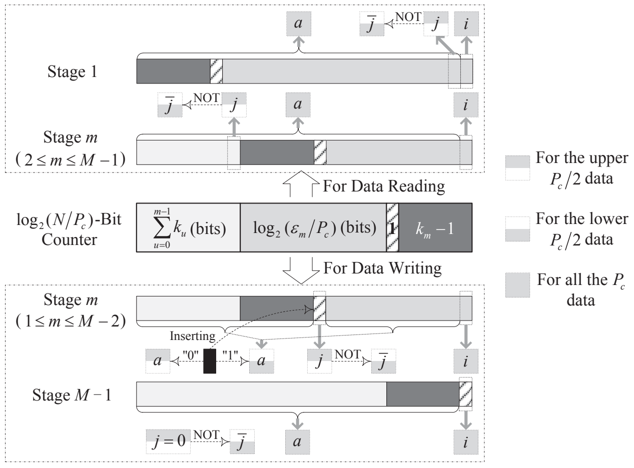
Stage 1
Stage m ($2 \le m \le M-1$)
For Data Reading
$\log_2(N/P_c)$-Bit Counter
$\sum_{u=0}^{m-1} k_u$ (bits)
$\log_2(\varepsilon_m/P_c)$ (bits)
k_m-1
For Data Writing
Stage m ($1 \le m \le M-2$)
Inserting
"0"
"1"
Stage $M-1$
NOT
$j=0$
For the upper $P_c/2$ data
For the lower $P_c/2$ data
For all the P_c data

Fig. 5. The way of producing data access parameters for stage 1 to $M-1$.

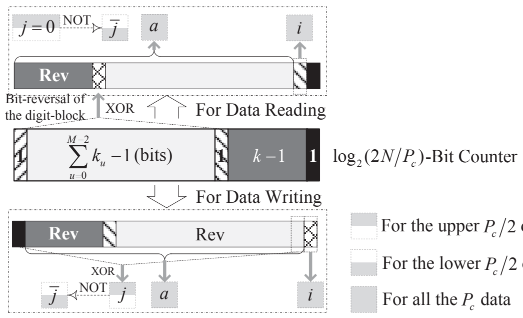
$j=0$
NOT
Rev
Bit-reversal of the digit-block
XOR
For Data Reading
$\sum_{u=0}^{M-2} k_u - 1$ (bits)
$k-1$
$\log_2(2N/P_c)$-Bit Counter
For Data Writing
Rev
For the upper $P_c/2$ data
For the lower $P_c/2$ data
For all the P_c data

Fig. 6. The way of producing data access parameters for stage M.

experience an additional delay of 2^{k-1} clock cycles based on the DSD component in DMU. The data writing adopts the mapping rule

$$i = \left\lfloor \frac{2d'}{P_c} \right\rfloor \bmod 2,\; j = \left(\left\lfloor \frac{2d'}{N} \right\rfloor + \left\lfloor \frac{d'}{P_c} \right\rfloor \right) \bmod 2, \\ a = \begin{cases} \left(\left\lfloor \frac{2d'}{N/P_c} \right\rfloor \bmod P_c \right) \frac{N}{2P_c^2} + \\ \mathrm{rev}_{n-2k-1}\left(\left\lfloor \frac{d'}{P_c} \right\rfloor \bmod \frac{N}{2P_c^2} \right), & N > 2^{2k} \\ \left\lfloor \frac{d'}{P_c} \right\rfloor \bmod \frac{N}{2P_c}, & N \in \{2^{2k}, 2^{2k-1}\} \end{cases} \tag{9}$$

where $d' = \mathrm{rev}_n(d)$. (9) meets **C1** for data writing because the concurrent samples have the same i, a. By considering the additional delay coming from DMU, the latency between continuous data reading and data writing reaches $3 \cdot 2^{k-1}$ clock cycles for the first $N/2$ samples, and 2^{k+1} clock cycles for the last $N/2$ ones, which can guarantee the satisfaction of **C2** and **C3** at stage M. Similar to the previous $M-1$ stages, the data access parameters for stage M can be attained using a $\log_2(2N/P_c)$-bit counter, as revealed from Fig. 6.

D. Providing FFT Output

The FFT results acquired from stage M are in bit-reversed order. To attain the in-order FFT output, $d' = \mathrm{rev}_n(d)$ in (9) should be replaced by d when it is used for data reading, which satisfies **C1** as the concurrent reading data are based on the

TABLE I
HARDWARE COMPLEXITY AND PERFORMANCE OF THE PROPOSED RADIX-2^k FFT PROCESSOR

PE Complexity			**Data Caching**		**Latency**† (*clock cycles*)	**Throughput**† (*clock rates*)
Complex adders (CA)	Complex mult. (CM)		RAMs (*words*)	Registers (*words*)	$\frac{N}{q}+\lceil\frac{n}{k}\rceil(2^{n-k}+2^k)+2^{k_1}+2^{k-1}$	$\begin{cases}1/(1/q+t) & \text{one DCM,}\\ 1/(1/q+\max\{t-1/q,0\}) & \text{ping-pang DCMs.}\end{cases}$ with $t=\lceil\frac{n}{k}\rceil(2^{-k}+2^{k-n})+2^{k_1-n}+2^{k-n-1}$
	Constant	General				
$k2^k$	$\approx k2^{k-3}$	2^k	$\frac{N}{4}\times 4$	2^{2k}		

† If $q=2^k$ and $N \gg 2^k$, then the minimum latency is $(1+\lceil n/k\rceil)\,2^{n-k}$ clock cycles, the maximum throughput is $2^k/(1+\lceil n/k\rceil)$ clock rates for one DCM, and $2^k/\lceil n/k\rceil$ clock rates for two DCMs operated in ping-pang mode.

TABLE II
COMPARISON OF 16384-POINT RADIX-2^4 FFT PROCESSOR WITH DIFFERENT HARDWARE SCHEMES (16-BIT WORDLENGTH)

	PE Cost†	**RAM Type**	**Memory Occupation**	**Throughput** (*clock rates*)
[1]	56/12/12	DP	128K×4+512K	4 (4/4)‡
[2]	64/16/16	DP	32K×16	3.2 (16/16)
[6]	64/8/16	SP	8K×64	2.67 (16/16)
[7]	32/8/8	SP	32K×16	1.6 (16/16)
[9]	64/8/16	SP	16K×32+8K	0.8 (2/16)
Proposed	64/16/16	SP	128K×4+8K	3.2 (16/16)

† 56/12/12 represents the number of CAs, constant CMs, and general CMs.
‡ 4/4 corresponds to the parameters $q_{\max}$ and P_c.

TABLE III
HARDWARE CONSUMPTION AND PERFORMANCE OF DIFFERENT FFT PROCESSORS IN FPGA ($N = 16384$, $k = 4$, 16-BIT WORDLENGTH)

	[2]	[9]	Proposed
Parallelism ($q_{\max}/P_c$)	16/16	2/16	16/16
Num. of Slice Registers	8864	9110	9016
Num. of Slice LUTs	7686	7125	7011
Num. of Occupied Slices	3192	3052	3014
Num. of DSP48Es	96	72	96
Block RAM (18K-bit)	32 (dual)	32 (single)	32 (single)
Mem. Access per FFT (bits)	5242880	5242880	5242880
Max. Freq. (MHz)	262.5	238.2	285.3
Min. Latency (us)	19.78	51.91	18.21
Max. Throughput (MS/s)	828.35	189.84	899.6

same j, a. Moreover, (9) becomes identical to (6) if further reversing the last $n-2k-1$ digits of a at $N > 2^{2k}$, and this allows the use of in-place data caching strategy, i.e., the overlap between caching input data using (6) and supplying FFT output using (9). In this case, the latency between data reading and data writing should keep to one or other odd clock cycles, so that **C2** and **C3** can be satisfied. In addition, for the odd-round (i.e., $1, 3, \cdots$) N-point FFT computation, all the mapping rules are followed exactly. While for the even-round (i.e., $2, 4, \cdots$) FFT computation, a in all the above mapping rules should be modified by reversing the last $n-2k-1$ digits before utilization.

IV. COMPARISON

The kernel metrics of the proposed radix-2^k FFT processor for FFT length $N = 2^n$, I/O parallelism q are listed in Table I. Under $N = 16384$, $k = 4$ and the maximum q ($q_{\max}$), the FFT processor based on different schemes are compared in Table II. MB solutions including [2], [6], [7], [9] and the proposed design deploy one DCM with $P_c = 16$, while the pipelined solution in [1] has $P_c = 4$. The proposed design and [2] have the best throughput capability among the MB FFT modules. However, the latter relies on dual-port RAMs and thus involves higher area occupation. For a 4-stage pipelined PE, [6] needs $4P_c$ separate RAM instances to gain conflict-free data access, while the enormous small RAMs will degrade the area efficiency. The use of [9] encounters parallelism bottleneck as the relevant $q_{\max} = 2$ compared to $q_{\max} = 16$ for other MB design. The pipelined-parallel FFT processor in [1] can further improve throughput at an expense of higher memory consumption.

The FFT processors based on different hardware schemes are tested on Xilinx Kintex 7 field programmable gate array (FPGA) XC7K325T using the design compiler Vivado 2015.2. The target FPGA has a speed grade of -3. Implementation results are listed in Table III. For the FFT module given by [2], physical address and bank index of each data should be separately computed. In the proposed design and [9], by contrast, the data access information is shared by the concurrent data, which allows a lower occupation of slices. Furthermore, the latter two schemes can be less area-consuming due to the use of single-port RAMs. While if considering latency and throughput, the low I/O parallelism ($q_{\max}$) will degrade the availability of [9]. The memory reading and writing account for the primary energy consumption of a MB FFT processor. Note that the tested schemes involve the same amount of memory access in computing an N-point FFT, it is suggested that the proposed design achieves similar performance as the competing schemes in energy efficiency.

V. CONCLUSION

In this paper, we have demonstrated the design of a radix-2^k FFT processor with computing parallelism $P_c = 2^k$ and the maximum I/O parallelism $q_{\max} = P_c$. The design was based on SP RAMs which are merged into 4 banks to support arbitrary P_c and I/O parallelism. Theoretical comparisons and FPGA testing results demonstrated the proposed design can outperform existing SP-RAM-based FFT design in throughput without degrading the performance in area occupation.

References

[1] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 23-32, Jan. 2013.

[2] 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.

[3] K.-F. Xia, B. Wu, T. Xiong and T.-C. Ye, "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 6, pp. 1919-1929, Jun. 2017.

[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, Reg. Papers*, vol. 59, no. 8, pp. 1752-1765, Aug. 2012.

[5] S.-N. Tang, F.-C. Jan, H.-W. Cheng, C.-K. Lin, and G.-Z. Wu, "Multimode memory-based FFT processor for wireless display FD-OCT medical systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 12, pp. 3394-3406, Dec. 2014.

[6] S. Richardson, D. Marković, A. Danowitz, J. Brunhaver, and M. Horowitz, "Building conflict-free FFT schedules," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1146-1155, Apr. 2015.

[7] S.-J. Huang and S.-G. Chen, "A high-parallelism memory-based FFT processor with high SQNR and novel addressing scheme," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2016, pp. 2671-2674.

[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, Exp. Briefs*, early access article, 2017.

[9] H.-F. Luo, Y.-J. Liu, and M.-D. Shieh,"Efficient memory-addressing algorithms for FFT processor design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2162-2172, Oct. 2015.