A Novel Conflict-Free Parallel Memory Access Scheme for FFT Processors

Qian-Jian Xing, Zhen-Guo Ma, and Ying-Ke Xu

***Abstract*—This brief presents a novel conflict-free access scheme for memory-based fast Fourier transform (FFT) processors. It is proved to satisfy the constraints of the mixed-radix, continuous-flow, parallel-processing, and variable-size FFT computations. An address generation unit is also designed and outperforms existing architectures with reduced gate delay and lower hardware complexity.**

***Index Terms*—Fast Fourier transform, memory-based architecture, conflict-free access scheme, continuous-flow.**

I. Introduction

FAST Fourier transform (FFT) plays a key role in the field of digital signal processing, encompassing a versatile range of applications such as spectral sensing, image processing, and wireless communication systems. As orthogonal-frequency-division multiplexing (OFDM) techniques have been widely used and various-size FFT operations in multi-standards is desired, a configurable, high-throughput and hardware-efficient FFT processor is one of the keys to develop a cognitive radio system. Up to present, numerous FFT processors have been designed for different applications and can be roughly divided into two categories: pipeline and memory-based architectures. The pipeline architecture usually leads to high throughput, but consumes relatively larger area and more power. On the contrary, the memory-based architecture is favored with only one set of the arithmetic processing unit and can easily vary the FFT sizes by simply changing the control mechanism. By utilizing high-radix arithmetic processing units in parallel, and driving the operating frequency to multiple times of the system sampling rate, the memory-architecture can meet the demand of continuous-flow and real-time processing. To minimize the requirement of storages to $2N$ words, a well designed memory addressing scheme becomes essential. Radhouane *et al.* [1] proposed a continuous-flow FFT processor having only two N-word memories based on the radix-2 algorithm. Johnson [2] uses an in-place strategy to produce conflict free mapping for radix-R FFT computations with single butterfly. Ma [3] reduces radix-2 FFT's address generation time at the expense of additional registers in the commutator to avoid the conflicting write. Hsiao *et al.* [4] gives the index mapping method for the generalized sized mixed-radix FFT, but it does not support multiple butterflies. In [5], the complex modulo operations in Hsiao's design is eliminated. Baek and Choi [6] proposed a multiple radix-2 butterflies address generation scheme available for conflict-free data access, but it's only suitable to radix-2 algorithms. Although memory access techniques for multiple radix-R butterfly units processing in parallel are illustrated in [7], it requires distributed memory at each stage and uses a lookup table of $O(PR^2)$ bits to implement the address generation. Takala *et al.* [8] provides a more sophisticated mapping of banks for any radix-R with any butterfly units in variable-size FFT computations, but it has no formal proof and is only verified by simulation. Inspired by Takala's algorithm, Richardson *et al.* [9], [10] shows how to build conflict-free schedules for FFTs using single-port instead of dual-port memory but with more memory banks. For example, an FFT with 2 radix-4 butterflies and 3-deep pipelines will need 32 memory banks. In [11]–[13], the bank conflicts due to the mixed-radix algorithm were removed by interchanging storage locations of butterfly outputs, but the strategy only works for single radix-2/4 butterfly unit. In [14] and [15], a global in-place but internal reshuffling strategy was proposed for continuous-flow memory-based FFT processors which incorporate radix-$2/2^2/2^3$ multi-path delay commutator (MDC) units in parallel, but its row and bank address assignments for different FFT size are incoherent and need to be predetermined in a lookup table. Sorokin and Takala [16] proposed a conflict-free parallel access scheme for continuous flow radix-2/4 FFT computations with any power-of-two numbers of memory modules. It also entirely eliminates the need for a "rotation unit" in the mapping hardware.

In this brief, a conflict-free access scheme for memory-based FFT architectures is proposed, which supports mixed-radix, continuous-flow, parallel butterfly and variable-size FFT computations. The organization of this brief is as follows. Section II reviews the constant geometry FFT algorithm and the continuous flow parallel processing FFT architecture. The proposed scheme is conceived and proved in Section III. Section IV designs an address generation unit and compares the proposed scheme with recently published works. Conclusion of the brief is drawn in Section V.

Manuscript received January 9, 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 College of Biomedical Engineering and Instrument Science, Zhejiang University, Hangzhou 310027, China (e-mail: xingqianjian@zju.edu.cn; 850501@zju.edu.cn; yingkexu@zju.edu.cn).

Digital Object Identifier 10.1109/TCSII.2017.2683643

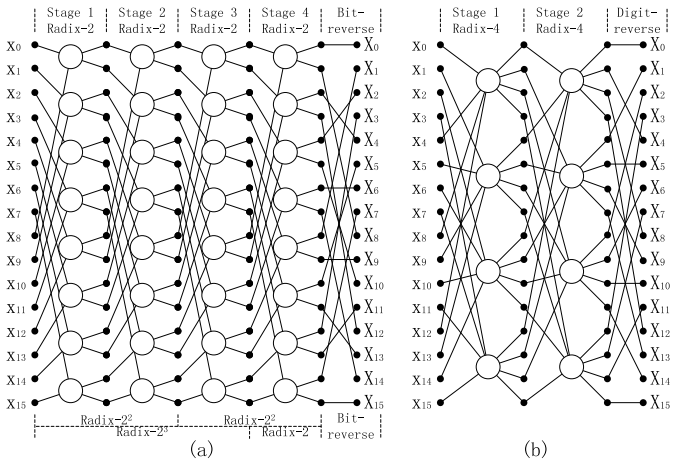

Fig. 1. The data flow graph of radix-2 and radix-4 constant geometry 16-point FFT.

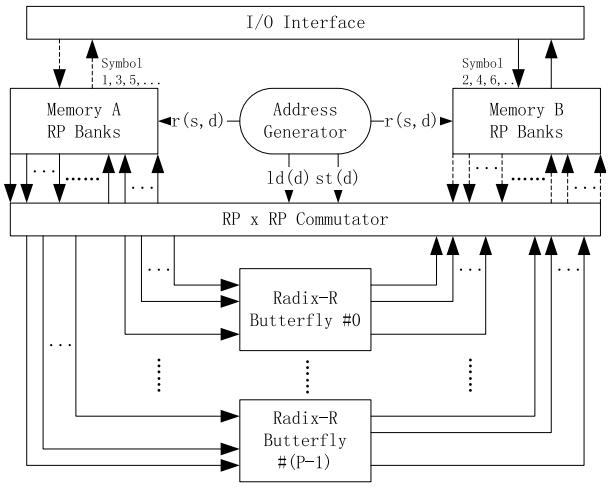

Fig. 2. Block diagram of a typical continuous-flow memory-based FFT architecture.

II. Continuous-Flow Parallel-Processing FFT Architecture

According to [17], the constant geometry N-point ($N = 2^n = R^m$, where m is an integer) radix-R ($R = 2^q, q \geq 1$) FFT can be defined as follows:

$$F_N = \prod_{i=0}^{m-1}\left(I_{R^i} \bigotimes P_{R^{m-i}}\right) \prod_{i=0}^{m-1}\left(I_{R^i} \bigotimes D_{R^{m-i}}\right)\left(I_{R^{m-1}} \bigotimes F_R\right)P_{R^m} \quad (1)$$

$$P_{R^{m-i}} = \left[e_0, e_R, \ldots, e_{R^{m-i}-R}, e_1, e_{R+1}, \ldots, e_{R^{m-i}-(R-1)}, \ldots, e_{R-1}, e_{2R-1}, \ldots, e_{R^{m-i}-1}\right] \quad (2)$$

$$D_{R^{m-i}} = diag\left\{W_N^{R^i(\beta \otimes l)}\right\} \quad (3)$$

where $\bigotimes$ is the Kronecker product operation; $W_N = e^{-2j\pi/N}$; $\beta = [0, 1, \ldots, R^{m-i-1} - 1], l = [0, 1, \ldots, R-1]$; e_i is the i^{th} column vector of the identity matrix I_{R^n}; F_R is the R-point Fourier transform matrix.

A typical memory-based continuous-flow FFT architecture is shown in Fig. 2. The two memory groups work in ping-pong fashion and each is switched between the computation mode and the concurrent I/O mode in turn. Each memory group is partitioned into RP banks if P ($P = R^p = 2^{pq}, p \geq 0$) butterflies are utilized to achieve high-throughput. In the computation mode, the butterfly outputs are written back to the same memory locations from which the operands are read, which is called in-place strategy. In the I/O mode, every time-domain input sample $x(i)$ of the next symbol is stored in the place where the frequency-domain output $X(k)$ of the current symbol has just been read out. If the input data of the current symbol is stored in natural order (NAT), the memory must be access in bit/digit reverse order (REV) for the next symbol and vice versa. Thus a symmetric reversal address map should be conceived to facilitate continuous-flow computation.

III. Conflict-Free Access Scheme

In this section, a new conflict-free memory addressing scheme is proposed based on the constant geometry FFT algorithm. When computing a radix-R N-point FFT in parallel with P butterflies, RP operands should be loaded from RP different memory banks in every cycle, and RP results should be distributed over the different memory banks as well, otherwise the memory conflicts will occur. Let $a = (a_{n-1}, \ldots, a_1, a_0) = \sum_{i=0}^{n-1} a_i 2^i$ represents the index of each element $x(i)$, then the mapped $(n-b)$-bit row address $r(a)$ and b-bit memory bank address $m(a)$ are defined as follows:

$$r(a) = \sum_{i=0}^{n-b-1} a_{i+b}2^i \quad (4)$$

$$m(a) = \sum_{i=0}^{b-1}(a_{i+n-b} \oplus a_i)2^i. \quad (5)$$

where $b = q(p+1)$, $n \geq 2b$, and $\oplus$ denotes bit-wise XOR operation. An example of memory bank address generation and contents of memory banks for 32-point radix-2 FFT with $q = 1$ and $p = 1$ is illustrated in Fig. 3. As shown in Fig. 1, the constant-geometry FFT algorithm doesn't have the in-place property directly, and the memory size needed for computation should to be $2N$ words in a single memory group. By interchanging storage locations of butterfly outputs according to the above index mapping method, the outputs of butterflies are written back to their input locations, and memory bank conflicts can be avoided in the FFT computations. Moreover, the Eq. (5) has an symmetric form for straight and reversed address representation, which can support concurrent I/O. For example, the memory bank address $m(a)$ is $(a_4 \oplus a_1, a_3 \oplus a_0)$ in Fig. 3 for straight address representation and $(a_0 \oplus a_3, a_1 \oplus a_4)$ for reversed address representation relatively.

From Eq. (1) and Eq. (2), we know that the permutation matrix P_{R^m} is independent of the stages and the address exchanging rule of the data for all stages can be written

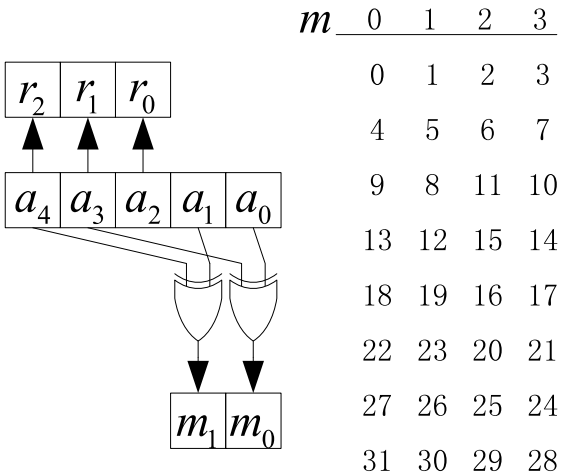

Fig. 3. Memory bank address generation and contents of memory for 32-point radix-2 FFT with 2 butterflies.

as follows:

$$P_{R^m}:\sum_{k=0}^{m-1}R^k w_k \rightarrow \sum_{k=0}^{m-2} w_k R^{k+1} + w_{m-1}, \tag{6}$$

where $w_k = \sum_{i=0}^{q-1} a_{i+kq}2^i$ since $R = 2^q$. The Eq. (6) can be further simplified as:

$$P_{R^m}:\sum_{i=0}^{n-1}2^i a_i \rightarrow \sum_{i=0}^{n-q-1} a_i 2^{i+q} + \sum_{i=0}^{q-1} a_{i+n-q}2^i. \tag{7}$$

Thus, we only need to study one stage where access conflicts can be avoided. Define $LD[j,i]$ to be the index of the ith operand for the jth butterfly in the cycle c of execution and $ST[j,i]$ to be the index of the ith result of the jth butterfly:

$$LD\big[j,i\big] = c*2^{pq} + j + i*2^{n-q} \tag{8}$$

$$ST\big[j,i\big] = c*2^{b} + j*2^{q} + i. \tag{9}$$

where $i = 0,1,\ldots,2^q-1$, $j = 0,1,\ldots,2^{pq}-1$, and $c = (c_{n-b-1},\ldots,c_1,c_0) = \sum_{k=0}^{n-b-1} c_k 2^k$. To avoid an access conflict, the following conditions should be satisfied when $i_1 \neq i_2$ or $j_1 \neq j_2$:

$$m\big(LD\big[j_1,i_1\big]\big) \neq m\big(LD\big[j_2,i_2\big]\big). \tag{10}$$

$$m\big(ST\big[j_1,i_1\big]\big) \neq m\big(ST\big[j_2,i_2\big]\big) \tag{11}$$

For storing, the expression $ST[j,i]$ can be divided into two parts:

$$STL\big[j,i\big] = j*2^q + i \tag{12}$$

$$STU\big[j,i\big] = c*2^b. \tag{13}$$

In each cycle, $STL[j,i]$ is varied and just covers the 2^b memory banks, while $STU[j,i]$ is constant for any j,i. Thus $m(ST[j,i])$ is determined by $STL[j,i]$ since $STU[j,i] \geq 2^b$ and $STL[j,i] < 2^b$, and Eq. (11) can be met obviously.

For loading, since i only acts upon the most significant q bits of the operands $LD[j,i]$ and j just works for the least significant pq bits, we divide $LD[j,i]$ into double parts and allocate the LSBs of c into $LDU[j,i]$ and the MSBs of c into $LDL[j,i]$ as there's an $(n-b)$-bit gap between the two operands of the bit-wise XOR operation in Eq. (5):

$$LDL\big[j,i\big] = \sum_{k=q}^{n-b-1} c_k 2^{pq+k} + j \tag{14}$$

$$LDU\big[j,i\big] = i2^{n-q} + \sum_{k=0}^{q-1} c_k 2^{pq+k} \tag{15}$$

By substituting Eq. (14) and Eq. (15) into Eq. (5) separately, $m([LD[j,i])$ can be solved as:

$$m\big(\big[LD\big[j,i\big]\big) = m\big(\big[LDL\big[j,i\big]\big]\big) + m\big(\big[LDU\big[j,i\big]\big]\big)2^{pq}. \tag{16}$$

That is the lower pq bits of the $m([LD[j,i])$ are determined by $LDL[j,i]$ and the upper q bits by $LDU[j,i]$. Similarly to the storing analysis, $m(LDL[j,i_1]) \neq m(LDL[j,i_2])$ if $i_1 \neq i_2$ and $m(LDU[j_1,i]) \neq m(LDU[j_2,i])$ if $j_1 \neq j_2$. That means there's no loading memory bank address overlap for any j and i pairs.

After $log_R N$ stages, the computed FFT symbol should be exported in bit/digit reverse order with $ar = \sum_{i=0}^{n-1} a_{\lfloor (n-1-i)/q \rfloor q + i\%q} 2^i$, and the corresponding memory bank address can be expressed as:

$$\begin{aligned} m(ar) &= \sum_{i=0}^{b-1} 2^i (ar_{i+n-b} \oplus ar_i) \\ &= \sum_{i=0}^{b-1} 2^i \big(a_{\lfloor (b-1-i)/q \rfloor q + i\%q} \oplus a_{\lfloor (b-1-i)/q \rfloor q + i\%q + n - b}\big) \end{aligned} \tag{17}$$

where % denotes the modulo operation. As the index union $\cup_{i=0}^{b-1}(\lfloor (b-1-i)/q \rfloor q + i\%q)$ is just a shuffle of the vector $[0,1,\ldots,b-1]$, it's easy to conclude that Eq. (10) and Eq. (11) are valid for digit reversed order sequence. That means the smooth read-out and write-in memory access without conflicts can be guaranteed for all the stages, which ensures continuous-flow processing.

When going into implementation, the row address $r(s,d)$ can be formulated with the cycle counter c and the memory bank index $d = (d_{b-1},\ldots,d_1,d_0) = \sum_{k=0}^{b-1} d_k 2^k$. Substituting Eq. (4) into Eq. (7), the row address $r(s,b)$ for each stage $s = 1,2,\ldots,n/q$ satisfies:

$$r(1,d) = \sum_{i=0}^{n-b-1} c_{(i+q)\%(n-b)} 2^i \oplus \sum_{i=b-q}^{b-1} d_i 2^{i\%(n-b)} \tag{18}$$

$$r(s+1,d) = \sum_{i=0}^{n-b-1} r(s,d)_{(i+q)\%(n-b)} 2^i \oplus \sum_{i=b-q}^{b-1} d_i 2^i. \tag{19}$$

Then the row address $r(s,d)$ can be solved as follows:

$$\begin{aligned} &r(s,d) \\ &\quad = \sum_{i=0}^{n-b-1} c_{(i+qs)\%(n-b)} 2^i \oplus \left(\oplus_{j=0}^{s-1} \sum_{i=b-q}^{b-1} d_i 2^{(i-jq)\%(n-b)} \right). \end{aligned} \tag{20}$$

where $\oplus_{j=0}^{s-1}$ is an iterative bit-xor operation of the memory bank index d of all stages with j from 0 to $s-1$ and it can

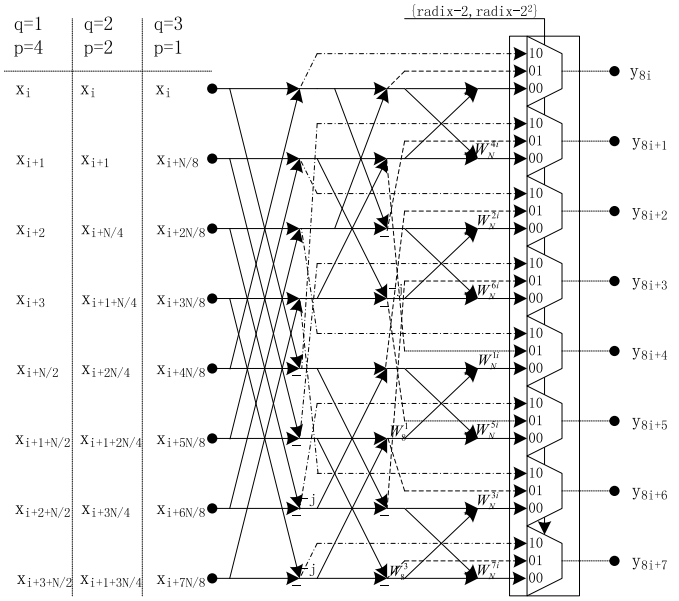

Fig. 4. Radix-$2/2^2/2^3$ butterfly unit.

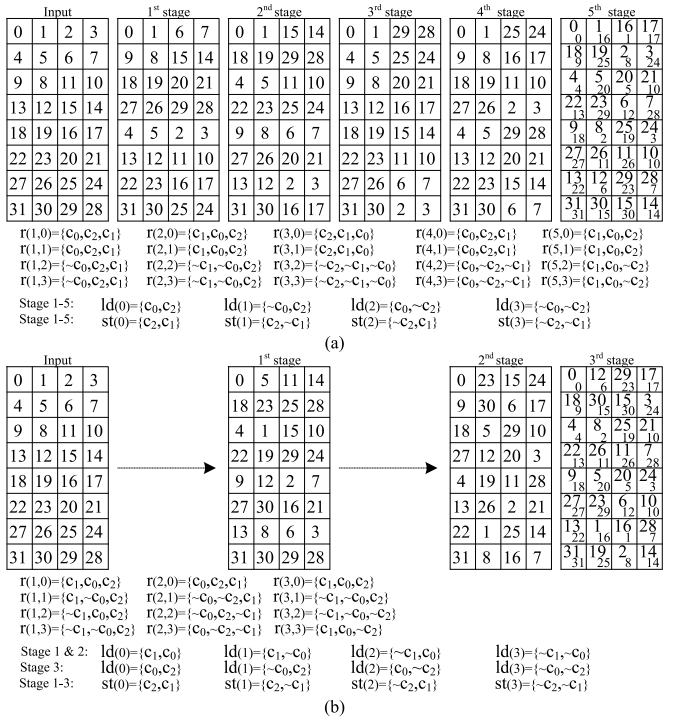

Fig. 5. Access scheme for 32-point FFT with 4 memory banks: (a) radix-2 (b) radix-$2/2^2$. The symbol $\sim$ represents the inverse of c_i.

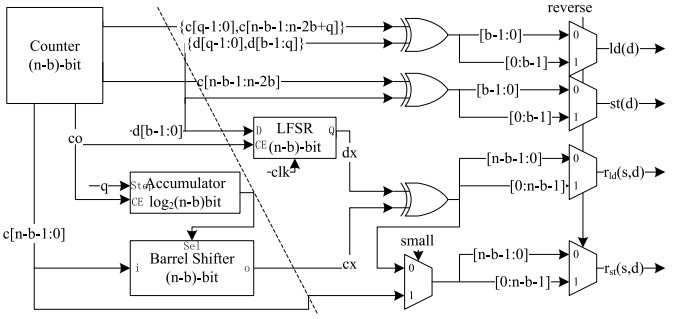

Fig. 6. Bank and row address generation unit.

be implemented by a linear feedback shift register (LFSR) as shown in Fig. 6.

Despite of the row address to read/write the data from/into the memory banks cycle by cycle, the commutator's control signals $ld(d)/st(d)$ for stride permutation in Fig. 2 can also be derived from Eq. (5) and Eq. (7):

$$ld(d) = \sum_{i=0}^{q-1}(c_i \oplus d_i)2^{b-q+i} + \sum_{i=0}^{b-q-1}\left(c_{n-2b+q+i} \oplus d_{i+q}\right)2^i \tag{21}$$

$$st(d) = \sum_{i=0}^{b-1}(c_{n-2b+i} \oplus d_i)2^i. \tag{22}$$

As the parameter b is a function of two variables, different combinations of q and p can produce the same $m(a)$, such as $q = 1, p = 1$ and $q = 2, p = 0$. Therefore, by merging q stages in Fig. 1(a) and applying the radix-2^q butterfly in Fig. 4, our proposal can support the mixed-radix algorithm. For example, a 32-point FFT can be decomposed into $2 \times 2 \times 2 \times 2 \times 2$, $2^2 \times 2^2 \times 2$ or $2^3 \times 2^2$. The access scheme for 32-point FFT with radix-2 and radix-$2/2^2$ is shown in Fig. 5, illustrating how the operands are arranged in 4 memory banks at each stage, as well as the row address $r(s, d)$ and the commutator's control signals $ld(d)/st(d)$ calculated from Eq. (20) to Eq. (22). The digit at the right-bottom corner in the last stage is the index of the bit-reversed output, which is necessary to export the frequency-domain output $X(k)$ in natural order and import the next time-domain symbol $x(i)$ in bit/digit reverse order.

IV. Address Generator and Comparison

The address generator is a key module of the FFT processor in Fig. 2, and it can be designed to support various-size FFT computation with low complexity based on the access scheme in Section III. As shown in Fig. 6, the address generator can be split into two parts by the dashed line. The left part is shared by all of the memory banks while the right part should be instantiated for each memory bank with individual index d. The bank address can be computed directly by a unary function of $(n-b)$-bit counter c and the bank index d. The row address can be interpreted as follows. Firstly, a $(log_2 n - b)$-bit accumulator is required to produce the number of positions by which the bus $c[n-b-1:0]$ should be rotated. The accumulator doesn't add the radix q_s until the current stage's computation completes and the counter overflows with the carry-out co asserted. Secondly, an $(n-b)$-bit barrel shifter rotates the input c and reflect the shifted data inputs on the output cx. Thirdly, the row address is obtained by performing bit-wise XOR operation between the vectors dx and cx. Each time the counter overflows, the dx is updated with an $(n-b)$-bit LFSR when the current stage's computation completes. Finally, the signal "reverse" controls the multiplexers to export the $ld(d)/st(d)/r(s,d)$ in forward/reversed address representation, which is adapted to the NAT/REV sequence.

TABLE I
COMPARISON OF DIFFERENT CONFLICT-FREE ADDRESSING SCHEMES

Features	[2] [3]	[8] [10]	[12] [13]	[4] [5]	[16]	[14] [15]	Proposed
Radix-	R	R	All	$2/4$	2/4	2^q	R, 2^q
Parallel Butterfly	No	Yes	No	No	Yes	Yes	Yes
In-place	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Memory Size	$2N$	$2N$	$2N$	$2N$	2N	$2N$	$2N$
Continuous Flow	Yes	Yes	Yes	Yes	Yes	Yes	Yes

[1] The parameter P must be power of R in our proposed scheme to keep the symmetry of addressing.

TABLE II
HARDWARE COMPLEXITY AND DELAY COMPARISON OF THE BANK ADDRESS GENERATION

Design Schemes	Components	Gate Delay
[2] [6] [12] [13]	$2(n-1)$ 1-bit Multiplexers $(n/2-1)$ 2-bit Adders	$2T_{and}+$ $4T_{and}\lceil log_2 n-1\rceil$
[14] [15]	bn 2-input AND gates b n-input XORs $bnlog_2 n$-bits ROM	$T_{and}+$ $2T_{and}\lceil log_2 n\rceil$
[8]	$(b-1)$ 2-input AND gates 1 b-bit barrel shifter b $\lceil n/b\rceil$-input XORs $b(b-1)log_2 n$-bits ROM	$2T_{and}+$ $T_{and}\lceil log_2 b\rceil+$ $2T_{and}\lceil log_2(n/b)\rceil$
[9] [10] [16]	b $\lceil n/b\rceil$-input XORs	$2T_{and}\lceil log_2(n/b)\rceil$
Proposed	b 2-input XORs	$2T_{and}$

When the FFT size is smaller than 2^n-point, the accumulator and the LFSR stop running since the carry-out *co* never goes high. The signal "small" is asserted to logic "1", and the row address for storing is equal to c. Each memory bank is marked off into two segmentations, and the operands are read from the lower half while the results are written into the upper and vice versa at next stage.

As shown in Table I, our scheme can support all of the features at the same time. It fulfills the continuous-flow requirements with minimum storage of 2*N* words, and supports mixed-radix algorithms as well as parallel processing. Compared with [16], our proposal also supports the multi-bank addressing for higher radix to decrease computation cycles. Different from [14]–[16], the proposed index-to-bank mapping method is consistent for fixed and mixed radix, only determined by the number of banks, which is helpful to reduce the complexity of the address generation circuit.

Assuming the delays of the XOR/Multiplexer operation to be $2T_{and}$, an n-bit barrel shifter to be $T_{and}\lceil log_2 n\rceil+T_{and}$ and an n-bit carry lookahead adder to be $2T_{and}\lfloor log_2(n-1)\rfloor+4T_{and}$, the hardware complexity and gate delay comparisons of the schemes are listed in Table II. Obviously, the presented bank address generator is independent of the size of FFT and requires simply 2-input XOR/Multipliexer operations for various-size FFT computation. Therefore it outperforms other techniques with both the delay and hardware complexity.

Moreover, the row address can be produced by only an additional accumulator and a barrel shifter, which is shared by all the memory banks. The gate delay of the row address generator equals to $T_{and}\lceil log_2(n-1)\rceil+7T_{and}$ and is lower than Johnson's design [2] when the radix is 2. On the contrary, there are additional steps including bit insertion for row address assignment in [14]–[16] and the address mapping for respective FFT size is irregular, both of which increase the circuits' complexity.

V. CONCLUSION

In this brief, a conflict-free addressing scheme for constant geometry FFT computation was presented. It was proved that the constraints of mixed-radix, parallel processing, continuous-flow and minimum storage can be satisfied simultaneously. Compared with previous works, it is evident that our proposal has obvious advantages of flexibility and efficiency in terms of gate delay and hardware complexity.

REFERENCES

[1] 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 21st Century Int. Symp. Circuits Syst. Emerg. Technol. (IEEE Cat No.00CH36353)*, vol. 1. Geneva, Switzerland, 2000, pp. 116–119.

[2] 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.

[3] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Trans. Signal Process.*, vol. 47, no. 3, pp. 907–911, Mar. 1999.

[4] 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.

[5] C. Ma, Y. Xie, H. Chen, Y. Deng, and W. Yan, "Simplified addressing scheme for mixed radix FFT algorithms," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Florence, Italy, May 2014, pp. 8355–8359.

[6] J. Baek and K. Choi, "New address generation scheme for memory-based FFT processor using multiple radix-2 butterflies," in *Proc. Int. SoC Design Conf.*, vol. 1. Busan, South Korea, Nov. 2008, pp. I-273–I-276.

[7] D. Reisis and N. Vlassopoulos, "Conflict-free parallel memory accessing techniques for FFT architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3438–3447, Dec. 2008.

[8] J. H. Takala, T. S. Jarvinen, and H. T. Sorokin, "Conflict-free parallel memory access scheme for FFT processors," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, vol. 4. Bangkok, Thailand, May 2003, pp. IV-524–IV-527.

[9] S. Richardson, O. Shacham, D. Markovic, and M. Horowitz, "An area-efficient minimum-time FFT schedule using single-ported memory," in *Proc. IFIP/IEEE 21st Int. Conf. Very Large Scale Integr.*, Istanbul, Turkey, 2013, pp. 39–44.

[10] 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.

[11] J. H. Baek, B. S. Son, B. G. Jo, M. H. Sunwoo, and S. K. Oh, "A continuous flow mixed-radix FFT architecture with an in-place algorithm," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, vol. 2. Bangkok, Thailand, May 2003, pp. II-133–II-136.

[12] 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.

[13] A. T. Jacobson, D. N. Truong, and B. M. Baas, "The design of a reconfigurable continuous-flow mixed-radix FFT processor," in *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 2009, pp. 1133–1136.

[14] 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.

[15] K. Xia, B. Wu, X. Zhou, and T. Xiong, "A generalized conflict-free address scheme for arbitrary 2^k-point memory-based FFT processors," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Montreal, QC, Canada, May 2016, pp. 2126–2129.

[16] H. Sorokin and J. Takala, "Conflict-free parallel access scheme for mixed-radix FFT supporting I/O permutations," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Prague, Czech Republic, May 2011, pp. 1709–1712.

[17] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," *J. ACM*, vol. 15, no. 2, pp. 252–264, Apr. 1968. [Online]. Available: http://doi.acm.org/10.1145/321450.321457