A Generalized Mixed-Radix Algorithm for Memory-Based FFT Processors

Chen-Fong Hsiao, Yuan Chen, *Member, IEEE*, and Chen-Yi Lee, *Member, IEEE*

***Abstract*—In this brief, a generalized mixed-radix (GMR) algorithm is proposed for memory-based fast Fourier transform (FFT) processors to support prime-sized and traditional 2^n-point FFTs simultaneously. It transforms the index to a multidimensional vector for efficient computation. By controlling the index vector to satisfy the "vector reverse" behavior, the GMR algorithm can support not only in-place policy for both computation and I/O data for continuous data flow to minimize the memory size but also multibank memory structures to increase the maximum throughput without memory conflict. Finally, a low-complexity implementation of an index vector generator is also proposed for our algorithm.**

***Index Terms*—Continuous data flow, fast Fourier transform (FFT), generalized mixed radix (GMR), in-place, vector reverse.**

I. Introduction

The orthogonal frequency-division multiplex (OFDM) modulation technique has been widely exploited in communication systems, such as 802.11, terrestrial digital video broadcasting (DVB-T), and terrestrial digital multimedia broadcasting (DMB-T). However, OFDM modulation involves the discrete Fourier transform (DFT) that needs substantial computation. Today, various FFT processors, such as pipelined or memory-based architectures, have been proposed for different applications. However, for long-size FFT processors, such as the 2048-point FFT, the pipelined architecture would cost more area and power than the memory-based design. Hence, memory-based approach has gained more and more attention recently in FFT processor designs for long-size DFT applications.

For the memory-based processor design, minimizing the necessary memory size is effective for area reduction since the memory costs a significant part of the processor. On the other hand, the FFT processor usually adopts on-chip static random access memory (SRAM) instead of external memory. The reason is the high-voltage I/O and the large capacitance in the printer-circuit-board (PCB) trace would increase power consumption for external memory. Besides the power issue, using external memory also increases the PCB-level verification cost for end-product manufacturers. Therefore, it is a trend to use the on-chip SRAM for FFT processors and to conduct FFT optimization for better system-level integration.

To minimize the necessary memory size, an in-place approach [1] is taken for both butterflies output and I/O data. That is, the output data of butterflies are written back to their original location during the computation time. Moreover, for the I/O data, the new input data $x[n]$ would be put in the location of the output data $X[n]$ of the previous FFT symbol. On the other hand, for the memory-based processor, the high-radix structure would be taken to increase the throughput to meet real-time requirements. However, when both in-place policy and high-radix structure are adopted simultaneously, it would result in memory conflict problem if the data have not been addressed properly. Until now, there have been several publications on this issue, the in-place approach for multibank memory structure, to optimize the processor design [1]–[3]. However, the approaches mentioned in [1] and [2] only work for fixed radix-r. The approach in [3] only works for radix-2/4. Hence, all of them could not be used for prime-number memory-based DFT processor design, and this still remains a challenge today. Note that a prime-number FFT has been proposed in recent communication standards, e.g., the 3780-point DFT in Chinese direct TV (DMB-T) [4] and the 1536-point DFT in Third Generation Partnership Project Long-Term Evolution [5]. Although, for the prime-number FFT computation, the zero-padding method can be taken to approximate it by computing some 2^n-point FFT, it has poor signal-to-quantization-noise-ratio performance than a prime-radix FFT. Therefore, to develop an approach for a multibank in-place addressing algorithm for general size, an FFT is necessary.

In this brief, a generalized mixed-radix (GMR) algorithm is proposed to optimize the memory-based FFT processor design. It supports not only in-place policy to minimize the necessary memory size for both butterflies output and I/O data but also multibank memory structure to increase its maximum throughput to satisfy more system applications without memory conflict. After the algorithm is introduced, we take the 3780-point FFT as an illustrative example. Finally, a low-complexity hardware implementation of an index vector generator is also proposed for our algorithm.

II. Proposed GMR Algorithm

A. Index Mapping

The definition of a DFT is $X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}$. The method to decompose a large-size DFT into several small-size DFTs has been proposed in [6]. Here, we take the following decomposition equation for the size $N = N_1N_2$:

$$\begin{cases} n=(N_2n_1+A_2n_2) \bmod N, & n_1,k_1=0,1,\ldots,N_1-1 \\ k=(B_1k_1+N_1k_2) \bmod N, & n_2,k_2=0,1,\ldots,N_2-1. \end{cases} \quad (1)$$

Equation (1) maps the indexes n and k to the index vectors (n_1, n_2) and (k_1, k_2) from one dimension $[0, N-1]$ into two dimensions $[0, N_1-1] \times [0, N_2-1]$. The selection of coefficients A_2 and B_1 depends on the relation between N_1 and

Manuscript received January 4, 2009; revised April 5, 2009, June 27, 2009, and September 9, 2009; accepted October 17, 2009. First published January 8, 2010; current version published January 15, 2010. This work was supported by the National Science Council of Taiwan under Grant NSC97-2221-E-009-166-MY3. This paper was recommended by Associate Editor M. M. Mansour.

The authors are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: holder@si2lab.org; ychen@si2lab.org; cylee@si2lab.org).

Digital Object Identifier 10.1109/TCSII.2009.2037262

N_2. In our approach, the coefficients for (1) are employed as follows.

Case 1) If N_1 and N_2 are relatively prime, we take that A_2 and B_1 satisfy

$$\begin{aligned} A_2 &= p_1 N_1 \text{ and } A_2 = q_1 N_2 + 1 \\ B_1 &= p_2 N_2 \text{ and } B_1 = q_2 N_1 + 1. \end{aligned}$$

Here, p_1, q_1, p_2, and q_2 are all positive integers. Then, the definition of the DFT could be rewritten as

$$\begin{aligned} X[k_1,k_2] &= \sum_{n_2}\sum_{n_1} x[n_1,n_2] W_{N_1}^{n_1k_1} W_{N_2}^{n_2k_2} \\ &= \sum_{n_2}\left\{\sum_{n_1} x[n_1,n_2] W_{N_1}^{n_1k_1}\right\} W_{N_2}^{n_2k_2} \\ &= \sum_{n_2} y[k_1,n_2] W_{N_2}^{n_2k_2}. \end{aligned} \tag{2}$$

Case 2) If N_1 and N_2 are not relatively prime, we take that $A_2 = B_1 = 1$. Then, the definition of the DFT could be rewritten as

$$\begin{aligned} X[k_1,k_2] &= \sum_{n_2}\sum_{n_1} x[n_1,n_2] W_{N_1}^{n_1k_1} W_{N_2}^{n_2k_2} W_N^{n_2k_1} \\ &= \sum_{n_2}\left\{W_N^{n_2k_1}\sum_{n_1} x[n_1,n_2] W_{N_1}^{n_1k_1}\right\} W_{N_2}^{n_2k_2} \\ &= \sum_{n_2} W_N^{n_2k_1} y[k_1,n_2] W_{N_2}^{n_2k_2}. \end{aligned} \tag{3}$$

Equations (2) and (3) imply that a larger size N-point FFT could be computed by two smaller sized N_1-point and N_2-point FFTs in the first and second stages, respectively. For the fixed n_2, the input data of the N_1-point FFT in the first stage are $x[n_1, n_2]$, $n_1 = 0, 1, \ldots, N_1 - 1$. Furthermore, the output data are $y[k_1, n_2]$, $k_1 = 0, 1, \ldots, N_1 - 1$. For the fixed k_1, the input data of the N_2-point FFT in the second stage are $y[k_1, n_2]$, $n_2 = 0, 1, \ldots, N_2 - 1$. Moreover, the output data are $X[k_1, k_2]$, $k_2 = 0, 1, \ldots, N_2 - 1$.

B. Bank Selection for Conflict-Free Processing

To minimize the memory size, the in-place policy [1] should be taken as described earlier. Here, we illustrate how to achieve conflict-free memory accesses when the in-place policy is adopted during the FFT computation. We distribute the input data into different memory banks.

Assume that the FFT size $N = N_1N_2N_3$, $N_3 \geqq N_1$, $N_3 \geqq N_2$, and the memory bank number is chosen to be N_3 here. The bank number could also be chosen as N_1 or N_2, and which one would be more suitable depends on the system requirement. Note that if the bank number is chosen as N_1 (or N_2), the following procedure can guarantee to access m data for m-point FFT in $\lceil m/N_1 \rceil$ clock cycles for $m = N_1$, N_2, and N_3.

Now, we compute the N_1-point FFT in the first stage, the N_2-point FFT in the second stage, and the N_3-point FFT in the third stage according to the following decomposition equations:

$$\begin{cases} n = (N_2N_3n_1 + A_1\tilde{n}_2) \bmod N, & n_1, k_1 = 0, 1, \ldots, N_1 - 1 \\ k = (B_1k_1 + N_1\tilde{k}_2) \bmod N, & \tilde{n}_2, \tilde{k}_2 = 0, 1, \ldots, N_2N_3 - 1 \end{cases} \tag{4}$$

$$\begin{cases} \tilde{n}_2 = (N_3n_2 + A_2n_3) \bmod N_2N_3, & n_2, k_2 = 0, 1, \ldots, N_2 - 1 \\ \tilde{k}_2 = (B_2k_2 + N_2k_3) \bmod N_2N_3, & n_3, k_3 = 0, 1, \ldots, N_3 - 1. \end{cases} \tag{5}$$

The computation order here is taken for illustration, and the other order can also be adopted. The coefficients A_1, B_1, A_2, and B_2 are employed as described in Section II-A.

The input data $x[n_1, \tilde{n}_2]$, $n_1 = 0, 1, \ldots, N_1 - 1$, of each N_1-point FFT in the first stage correspond to the same $\tilde{n}_2$. Since n_1 is $0, 1, \ldots, N_1 - 1$ and $N_3 \geqq N_1$, (6) would map different n_1 to a different $bank$ for each N_1-point FFT [1], i.e.,

$$bank = (N_1 + x) \bmod N_3. \tag{6}$$

Therefore, conflict-free memory access in the first stage could be achieved by (6). Here, x in (6) is an arbitrary constant number.

After the first stage, the N-point FFT has been divided into N_1 smaller size N_2N_3-point FFTs with index $(k_1, \tilde{n}_2)$, $k_1 = 0, 1, \ldots, N_1 - 1$. Note that, for each N_2N_3-point FFT, they have the same index k_1.

Similarly, by (5), we could also decompose the N_2N_3-point FFT into N_2-point and N_3-point FFTs and map the index from $(k_1, \tilde{n}_2)$ into (k_1, n_2, n_3). Since the data of each N_2N_3-point FFT have the same index k_1, similar to the discussion of (6), conflict-free memory access could also be achieved when we compute the N_2-point and N_3-point FFTs by (7) and (8), respectively, i.e.,

$$bank = (n_2 + x) \bmod N_3 \tag{7}$$

$$bank = (n_3 + x) \bmod N_3. \tag{8}$$

Combining (6)–(8), we get

$$bank = (n_1 + n_2 + n_3) \bmod N_3. \tag{9}$$

From the above discussion, the memory access conflict problem, which results from in-place policy during FFT computation, could be solved by distributing the input data into N_3 memory banks by (9).

Once the memory bank is selected, data need to be addressed. Any two data in the same memory bank should be mapped to a different address within the range from 0 to $N_1N_2 - 1$. For simplicity, we choose the following for data addressing:

$$address = N_2n_1 + n_2. \tag{10}$$

C. Concurrent I/O and Conflict-Free Processing

Since the in-place policy is adopted for I/O data, the new input data $x[i]$ should be put in the location of the output data $X[i]$ of the previously computed FFT symbol. However, in Section II-B, we have illustrated that our approach would put the new input data by (9) and (10). Hence, in the following, we will show how to satisfy (9) and (10) under the constraint that the new input data $x[i]$ should be put in the location of the output data $X[i]$ of the previously computed FFT symbol.

Suppose that $N = N_1N_2N_3$ and N_1, N_2, and N_3 are relatively prime. Then, there would have positive integers p_i and q_i for $i = 1, 2, 3$ such that

$$\begin{cases} p_1N_1 = q_1N_2N_3 + 1 \\ p_2N_2 = q_2N_1N_3 + 1 \\ p_3N_3 = q_3N_1N_2 + 1. \end{cases}$$

We observe the relation of the index mapping equation between the reverse decomposition orders.

Case 1) If we decompose N in the order N_1, N_2, and N_3. The decomposed equations for this decomposition order N_1, N_2, and N_3 could be obtained as follows:

$$\begin{cases} n = (N_2N_3n_1 + p_1N_1\tilde{n}_2) \bmod N \\ k = (p_2p_3N_2N_3k_1 + N_1\tilde{k}_2) \bmod N \end{cases} \tag{11}$$

$$\begin{cases} \tilde{n}_2 = (N_3n_2 + p_2N_2n_3) \bmod N_2N_3 \\ \tilde{k}_2 = (p_3N_3k_2 + N_2k_3) \bmod N_2N_3. \end{cases} \tag{12}$$

From (11) and (12), we get the index mapping equations for input and output data as follows for the decomposition order N_1, N_2, and N_3:

$$n = (N_2N_3n_1 + p_1N_1N_3n_2 + p_1p_2N_1N_2n_3) \bmod N \tag{13}$$
$$k = (p_2p_3N_2N_3k_1 + p_3N_1N_3k_2 + N_1N_2k_3) \bmod N. \tag{14}$$

Case 2) If we decompose N in the order N_3, N_2, and N_1. Similar to case I, the index mapping equations for input and output data are as follows for this decomposition order:

$$n = (N_1N_2n_1 + p_3N_1N_3n_2 + p_2p_3N_2N_3n_3) \bmod N \tag{15}$$
$$k = (p_1p_2N_1N_2k_1 + p_1N_1N_3k_2 + N_2N_3k_3) \bmod N. \tag{16}$$

Comparing (13)–(16), we find that input equation (13) and output equation (16) are the same. Furthermore, input equation (15) and output equation (14) are also the same. This means that the in-place policy could be supported for both butterflies output and I/O data without memory access conflict if the FFT is computed with the reverse decomposition order of the previous FFT symbol.

Fig. 1 shows the data flow of the 60-point FFT by our algorithm. The columns $bank$ and $addr$ represent the corresponding memory bank and address, respectively. The columns n and k represent the index of the input and output data, respectively. Since $60 = 3 \times 4 \times 5$, we could compute it by exploiting 3-point, 4-point, and 5-point FFTs. The figure shows that the memory data access is conflict-free during the FFT computation. Moreover, by comparing the output index of the second FFT symbol with the input index of the first FFT symbol, we find that the data flow of the third FFT symbol will return to the state of the first FFT symbol, implying that (13)–(16) are perfectly matched.

III. Example of the 3780-Point FFT Design

Since $3780 = 4 \times 3 \times 3 \times 3 \times 5 \times 7$, we could compute it by computing the three-point, four-point, five-point, and seven-point FFTs. Here, the data are distributed into seven memory banks.

First, we take the decomposition order 4, 3, 3, 3, 5, and then 7. For the first stage (4-point FFT), the following decomposition equation is taken:

$$\begin{cases} n = (945n_1 + 2836\tilde{n}_2) \bmod 3780, & n_1, k_1 = 0, 1, 2, 3 \\ k = (945k_1 + 4\tilde{k}_2) \bmod 3780, & \tilde{n}_2, \tilde{k}_2 = 0, \ldots, 944. \end{cases} \tag{17}$$

It maps the index n to the vector $(n_1, \tilde{n}_2)$ from [0, 3779] to [0, 3] × [0, 944], as shown in Table I. The data in each row of Table I are the input data of each 4-point FFT. The input order is decided by the index n_1, as shown in Fig. 2.

After the first stage, the original data have been divided into four independent groups for four 945-point FFTs corresponding to $k_1 = 0, 1, 2,$ and 3, respectively.

Similarly, for the 945-point FFT, we could also decompose it in the order $945 = 3 \times 3 \times 3 \times 5 \times 7$ and map the index

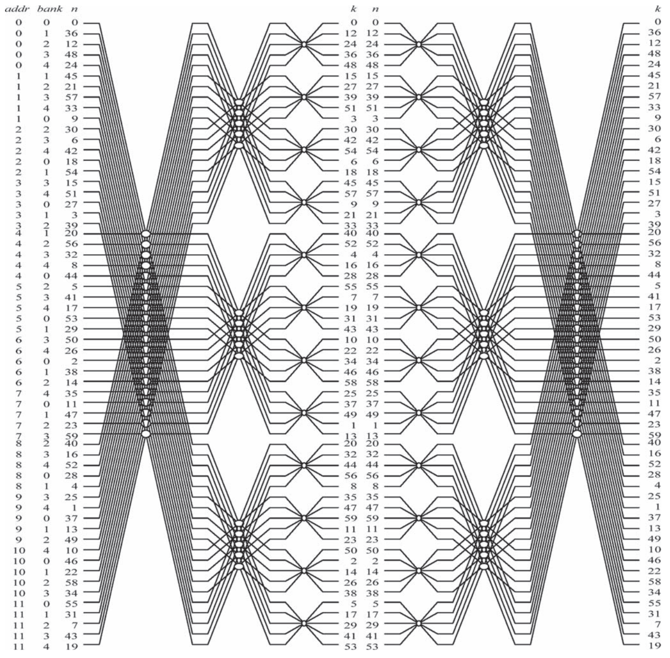

Fig. 1. Example of a 60-point FFT.

TABLE I
Index Mapping for the First Stage

	n_1=0	n_1=1	n_1=2	n_1=3
$\tilde{n}_2=0$	x[0]	x[945]	x[1890]	x[2835]
$\tilde{n}_2=1$	x[2836]	x[1]	x[946]	x[1891]
$\tilde{n}_2=2$	x[1892]	x[2837]	x[2]	x[947]
…	…	…	…	…
$\tilde{n}_2=944$	x[944]	x[1889]	x[2834]	x[3779]

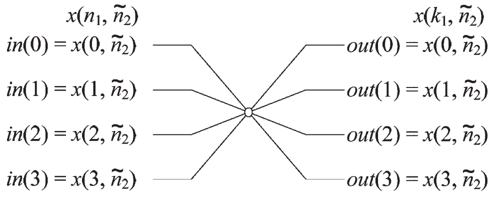
$x(n_1, \tilde{n}_2)$
$x(k_1, \tilde{n}_2)$
$in(0) = x(0, \tilde{n}_2)$
$out(0) = x(0, \tilde{n}_2)$
$in(1) = x(1, \tilde{n}_2)$
$out(1) = x(1, \tilde{n}_2)$
$in(2) = x(2, \tilde{n}_2)$
$out(2) = x(2, \tilde{n}_2)$
$in(3) = x(3, \tilde{n}_2)$
$out(3) = x(3, \tilde{n}_2)$

Fig. 2. Input and output data of a 4-point FFT.

$\tilde{n}_2$ to the vector $(n_2, n_3, n_4, n_5, n_6)$ from [0, 944] to $[0, 2] \times [0, 2] \times [0, 2] \times [0, 4] \times [0, 6]$. By combining all the decomposition equations for each stage, the full index mapping equations for the 3780-point FFT in this decomposition order are shown as follows:

$$\begin{aligned} n = {} & (945n_1 + 1260n_2 + 2940n_3 \\ & + 980n_4 + 1512n_5 + 540n_6) \bmod 3780 \end{aligned} \tag{18}$$
$$\begin{aligned} k = {} & (945k_1 + 2380k_2 + 3360k_3 \\ & + 2520k_4 + 2268k_5 + 540k_6) \bmod 3780. \end{aligned} \tag{19}$$

Similar to (9) and (10), we could also get the following full bank selection addressing equations:

$$bank = (n_1 + n_2 + n_3 + n_4 + n_5 + n_6) \bmod 7 \tag{20}$$
$$address = 135n_1 + 45n_2 + 15n_3 + 5n_4 + n_5. \tag{21}$$

During computation, we need to access r data from seven banks for r-point FFT computation simultaneously in each stage. From (20) and Fig. 2, when the bank of the first input data

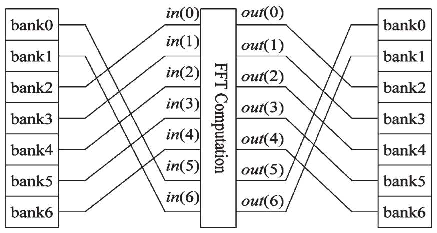

Fig. 3. Permutation of accessed data for $bank(0) = 2$.

$in(0)$, called $bank(0)$, is determined, the bank of other input data can also be determined as $bank(0) + 1 \bmod 7, bank(0) + 2 \bmod 7, \ldots$. Fig. 3 shows the permutation of accessed data for $bank(0) = 2$. The relation between the addresses of accessed r data is similar.

After the first FFT symbol, i.e., the odd FFT symbol, has been computed, all the indexes n_i will be transformed to k_i in (20) and (21). For the second input FFT symbol (i.e., the even FFT symbol), its mapping index, memory bank, and address can be determined by (19)–(21), respectively. This is because the even FFT input data $x_{\text{even}}[n]$ can be placed in the location of the odd FFT output data $X_{\text{odd}}[k]$ with $n = k$.

For the computation of the even FFT symbol, we take the reverse decomposition order of the odd FFT symbol, i.e., $3780 = 7 \times 5 \times 3 \times 3 \times 3 \times 4$. By similar approach, we map the input index n and the output index k to vectors $(a_1, a_2, a_3, a_4, a_5, a_6)$ and $(b_1, b_2, b_3, b_4, b_5, b_6)$, respectively, from [0, 3779] to $[0, 6] \times [0, 4] \times [0, 2] \times [0, 2] \times [0, 2] \times [0, 3]$ for the even FFT symbols. By Section II-C, the relations between n_i, k_i, a_i, and b_i are as follows. Note that the index vectors are reversely matched (vector reverse), i.e.,

$$\begin{aligned}(a_1, a_2, a_3, a_4, a_5, a_6) &= (k_6, k_5, k_4, k_3, k_2, k_1)\\ (n_1, n_2, n_3, n_4, n_5, n_6) &= (b_6, b_5, b_4, b_3, b_2, b_1).\end{aligned}$$

Then, the bank selection and the memory addressing for the even FFT symbol are as follows:

$$\begin{aligned}bank &= (a_1 + a_2 + a_3 + a_4 + a_5 + a_6) \bmod 7\\ address &= 135a_6 + 45a_5 + 15a_4 + 5a_3 + a_2.\end{aligned}$$

From Section II-B, the memory access would keep conflict-free for the even FFT symbol. Furthermore, from Section II-C, the data distribution of the third FFT symbol would return to the state of the odd FFT symbol. Then, a conflict-free general mixed-radix FFT processor can be achieved by reversing the decomposition order of the previous FFT symbol.

In this example, we compute the 3780-point FFT with six-stage small-size FFTs. Since the FFT sizes in different computation stages are different, making all memory banks keep full loading (being accessed simultaneously) during FFT computation is difficult. On the other hand, this computation needs 3780(data) $\times$ 6(stages) $\times$ 2(read/write) memory accesses. Since the memory bank number is 7, the minimum necessary clock rate for this processor is $3780 \times 6 \times 2/(3780 \times 7) = 1.72$. That is, it is impossible to finish the FFT computation with only a $1\times$ clock rate, and we need at least $2\times$ system clock to finish it.

IV. Implementation of the Index Vector Generator

The block diagram of the proposed memory-based DFT processor is shown in Fig. 4. It contains two N-word memory

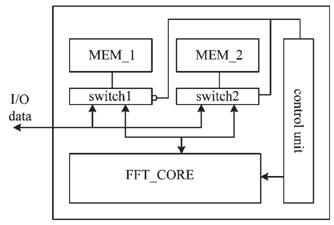

Fig. 4. Block diagram of the proposed FFT processor.

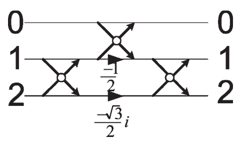

Fig. 5. Signal flow graph of a 3-point FFT.

modules, i.e., MEM_1 and MEM_2, to support both I/O data transfer and FFT computation via switch1 and switch2. Hence, we only need $2N$ words memory for real-time application.

The FFT_CORE is used to compute the decomposed small-size DFTs. Some algorithms to implement the butterfly engine within the FFT_CORE for 2^n-sized and prime-sized DFTs can be found in [3], [7], and [8]. Fig. 5 shows one method to implement the 3-point FFT [8]. This 3-point FFT can be finished by three 2-point FFTs. Similarly, other prime-radix FFT can also be achieved with the similar method. Hence, it is possible to compute the prime-radix FFT based on the 2-point FFT with low-complexity control logic to schedule the butterfly engine.

Generally, for a long-size DFT processor, the memory would almost dominate a full processor area [9]. For example, in the 4096-point FFT processor [9], memory and butterfly engine cost about 82% and 2% of the total area, respectively.

The control unit in Fig. 4 contains two index vector generators to generate the necessary index vectors for I/O data, respectively. These two generators are the same.

In the following, we introduce our proposed index vector generator used for I/O data. The FFT size $N = N_1N_2N_3$, which is discussed in Section II-B, is taken as an example. We let the coefficient $A_1 = qN_2N_3 + 1$ in (4). If N_1 and N_2N_3 are not relatively prime, then $q = 0$. Otherwise, q is some positive integer. Hence, the input equation in (4) can be rewritten as

$$n = (N_2N_3n'_1 + \tilde{n}_2) \bmod N. \tag{22}$$

Here, we define

$$n'_1 = (n_1 + q\tilde{n}_2) \bmod N_1 \tag{23}$$
$$n_1 = (n'_1 + q'\tilde{n}_2) \bmod N_1. \tag{24}$$

Similarly, the input equation in (5) can also be rewritten as

$$\tilde{n}_2 = (N_3n'_2 + n_3) \bmod N_2N_3 \tag{25}$$
$$n_2 = (n'_2 + r'n_3) \bmod N_2. \tag{26}$$

From (22) and (25), we have

$$n = (N_2N_3n'_1 + N_3n'_2 + n_3) \bmod N. \tag{27}$$

Equation (27) indicates that the transform from n to (n'_1, n'_2, n_3) could be realized by three accumulators. Hence, by combining (24), (26), and (27), the proposed index vector generator for I/O data could be designed as shown in Fig. 6. The ACC in Fig. 6 is an accumulator. The ACC n'_1, ACC n'_2, and

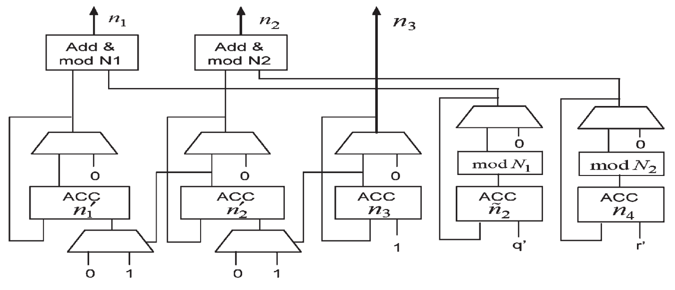

Fig. 6. Proposed index vector generator.

TABLE II
COMPARISON OF DIFFERENT ADDRESSING SCHEMES

	[10]	[11]	[1], [2], [3]	proposed
Multi-bank	Yes	No	Yes	Yes
In-place	No	Yes	Yes	Yes
Supported algorithm	Radix-2/4	Radix-2	Radix-2/4 or Radix-*r*	All radix
Memory	4*N* words	2*N* words	2*N* words	2*N* words

ACC n_3 make the counter that we have described above. Each of the modulo operation can be implemented by one comparator and one adder. Since the factor $q' = N_1 - q$ comes from the decompose equation of the first stage, the operation ACC $\tilde{n}_2$ should be set to zero when $n_2' = N_2 - 1$ and $n_3 = N_3 - 1$. Similarly, $r' = N_2 - r$ and the operation ACC n_4 should be set to zero when $n_3 = N_3 - 1$.

V. COMPARISON

The comparison among different addressing schemes is shown in Table II. Compared with the in-place policy, our work only costs half the memory size of the Jaguar II processor [10]. Our work can provide higher throughput and can be used in more system applications than [11] does. Moreover, since [1]–[3] only support fixed radix or radix-2/4, they could only work for the FFT size with the form $N = r^n$. Only our work could support several different radixes simultaneously such as the 3780-point FFT.

For the conventional 2^n-point FFT, Table III(a) shows the available minimum FFT computation cycles and the corresponding adopted algorithm for different addressing schemes [1]–[3]. Note that only our work can finish all the mentioned computation tasks within $2N$ clock cycles for the N-point FFT. For example, for the DVB-T/H system application, the 2048/4096/8192-point FFTs are needed, and our work is more suitable for this system than those of [1]–[3]. Table III(b) shows the area overhead that divides the memory from 1 block into 2/4/8 smaller memory banks. It is normalized to two-bank memory cost. From the table, the area overhead of the multibank would decrease as the FFT size increases. For the 8192/4096-point FFT application, the memory area of the eight-bank architecture only costs about 30% more than the two-bank structure and 20% more than the four-bank structure. Thus, for the applications with FFT size more than 4096, e.g., DVB-T/H, our work would be a good option since it can provide only about 20% area overhead than the four-bank structure with a 2× minimum necessary clock rate instead of a 4× minimum necessary clock rate for the four-bank structure.

TABLE III
(a) COMPUTATION CYCLES AND ADOPTED ALGORITHM FOR DIFFERENT ADDRESSING SCHEMES. (b) AREA COST OF MULTIBANK MEMORY

(a)

	[1], [2]		[3]		proposed	
	Cycle	Radix	Cycle	Radix	Cycle	Radix
1024-point	2560	4	2560	4	1024	2/8
2048-point	22528	2	6144	2/4	2048	4/8
4096-point	4096	8	12288	4	4096	8
8192-point	106496	2	28672	2/4	10240	2/8

(b)

	2-bank	4-bank	8-bank
1024-point	1	1.3	1.8
2048-point	1	1.2	1.5
4096-point	1	1.1	1.3
8192-point	1	1.1	1.3

VI. CONCLUSION

In this brief, a GMR algorithm has been proposed to optimize the general-size memory-based FFT processor design. It supports the in-place policy for both butterflies output and I/O data to minimize the necessary memory size. Hence, only $2N$ words memory is required for any size FFT computation for real-time requirements. Furthermore, our proposal also supports the multibank addressing for a high-radix structure without memory conflict by reversing the decomposition order of the previous FFT symbol. Finally, a low-complexity index vector generator has been proposed for our algorithm. It only costs a few accumulators, making our proposal very suitable for multistandard and multimode OFDM-based applications.

REFERENCES

[1] 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.

[2] J. A. Hidalgo, J. Lopez, F. Arguello, and E. L. Zapata, "Area-efficient architecture for fast Fourier transform," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 2, pp. 187–193, Feb. 1999.

[3] 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.

[4] Z.-X. Yang, Y.-P. Hu, C.-Y. Pan, and L. Yang, "Design of a 3780-point IFFT processor for TDS-OFDM," *IEEE Trans. Broadcast.*, vol. 48, no. 1, pp. 57–61, Mar. 2002.

[5] 3GPP TS 36.201 V8.3.0 LTE Physical Layer—General Description, E-UTRA, Mar. 2009.

[6] C. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-25, no. 3, pp. 239–242, Jun. 1977.

[7] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-25, no. 4, pp. 281–294, Aug. 1977.

[8] A. M. Despain, "Very fast Fourier transform algorithms hardware for implementation," *IEEE Trans. Comput.*, vol. C-28, no. 5, pp. 333–341, May 1979.

[9] C. L. Wey, S.-Y. Lin, and W. C. Tang, "Efficient memory-based FFT processors for OFDM applications," in *Proc. IEEE Electro/Inf. Technol.*, May 17–20, 2007, pp. 345–350.

[10] *Jaguar II Variable-Point (8-1024) FFT/IFFT*, Drey Enterprise Inc., Crosslake, MN, 1998.

[11] 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.*, May 2000, vol. 1, pp. 116–119.

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

[13] D. Reisis and N. Vlassopoulos, "Address generation techniques for conflict free parallel memory addressing in FFT architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3438–3447, Dec. 2008.