Optimum Circuits for Bit Reversal

Mario Garrido Gálvez, Jesus Grajal and Oscar Gustafsson

Mario Garrido Gálvez, Jesus Grajal and Oscar Gustafsson, Optimum Circuits for Bit Reversal, 2011, IEEE Transactions on Circuits and Systems - II - Express Briefs, (58), 10, 657-661.
http://dx.doi.org/10.1109/TCSII.2011.2164141
Postprint available at: Linköping University Electronic Press
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-71782

Optimum Circuits for Bit Reversal

Mario Garrido, *Member, IEEE*, Jesús Grajal and Oscar Gustafsson, *Senior Member, IEEE*

***Abstract*—This paper presents novel circuits for calculating the bit reversal on a series of data. The circuits are simple and consist of buffers and multiplexers connected in series. The circuits are optimum in two senses: they use the minimum number of registers that are necessary for calculating the bit reversal and have the minimum latency. This makes them very suitable for calculating the bit reversal of the output frequencies in hardware FFT architectures.**

The paper also proposes optimum solutions for reordering the output frequencies of the FFT when different common radices are used, including radix-2, radix-2^k, radix-4 and radix-8.

***Index Terms*—Bit Reversal, FFT, Pipelined Architecture**

I. Introduction

BIT REVERSAL is an algorithm that permutes a set of indexed data according to a reversing of the bits of the index [1]. This algorithm is used many times to sort out the output frequencies of the fast Fourier transform (FFT), which are usually provided in bit-reversed order.

The bit reversal algorithm has been studied for many years. Most approaches in the literature propose efficient procedures for calculating the bit reversal of a set of data stored in a memory with the aim of minimizing the computation time [2]–[5]. In some cases cache techniques are also considered [6], [7]. As these approaches carry out a procedure, they are suitable for calculating the bit reversal in computers or microprocessors.

For in-place FFT hardware architectures [8] memory-based approaches have also been proposed [9], [10]. They focus on the design of the address generator.

However, in pipelined FFT hardware architectures [11]–[14], samples are provided sequentially in a continuous flow. In this context the calculation of the bit reversal is a different problem from shuffling data in a memory. On the one hand, if samples are stored in a memory all input data are available from the beginning. This is not true for samples arriving in series. On the other hand, the bit reversal of a continuous flow of data also provides a continuous flow of data, instead of writing back the bit-reversed sequence in the memory. Currently, the bit reversal of a series of data is calculated using either double buffering [12], [15] or a single memory in which the memory address is generated in natural and bit-reversed order, alternatively for even and odd sequences [12], [14], [16].

This paper analyzes the problem of calculating the bit reversal of a series of data and presents the optimum solution to this problem. The solution consists of very simple circuits with buffers and multiplexers. These circuits are proven to be optimum in two senses. First, they require the minimum number of registers, i.e., the bit reversal cannot be calculated with fewer storage elements. Second, the circuits have the lowest latency that can be achieved. The theoretical minimum latency and number of registers for radix-2 were already discussed in [16]. However, simple circuits that use the lowest number of registers were not proposed.

Besides, when radices such as radix-4 or radix-8 are used, the outputs of the FFT are provided in an order different from bit reversal [10]. These cases are also studied and optimum circuits for these radices are obtained.

The presented circuits are derived by using the framework for mapping algorithms to architectures presented in [14]. This framework has already been used to derive circuits for matrix transposition and novel FFT hardware architectures [14], [17]. Circuits for bit reversal can also be obtained by using other approaches such as stride permutations [18]. However, to the best of the authors' knowledge no explicit results for bit reversal have been presented so far.

This paper is organized as follows. Section II briefly reviews the framework in [14]. Section III introduces the bit reversal algorithm and some consideration for calculating it in hardware. Section IV uses the framework to derive simple circuits for bit-dimension permutations on data arriving in series. Section V presents the optimum circuits for bit reversal and Section VI extends the results to different radices used in the FFT. Finally, some conclusions are drawn in Section VII.

II. Review of the Framework

This section summarizes some key ideas of the framework presented in [14], which is used to derive the circuits proposed in this paper. The framework links algorithms to their hardware architectures, showing the relationship between them. It is based on the theories of hypercubes [19] and bit-dimension permutations [20] and can be applied to algorithms that involve data management, such as matrix transposition or the FFT.

The framework considers an n-dimensional space with dimensions $x_{n-1} \dots x_1 x_0$. Each dimension can only take two binary values, i.e, $x_i \in \{0, 1\}, \forall i$. As the number of positions in the hyperspace is finite, they can be numbered. According to this, the position of a vector in the hyperspace is defined as:

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

where x_i is the value of the i-th component of the vector. It can be noted that this is equivalent to considering the components of the vector as a binary number. For instance, the position of

M. Garrido and O. Gustafsson are with the Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, e-mails: mariog@isy.liu.se, oscarg@isy.liu.se

J. Grajal is with the Department of Signal, Systems and Radiocommunications, Universidad Politécnica de Madrid, 28040 Madrid, Spain, e-mail: jesus@gmr.ssr.upm.es

This work was supported by the FPU Fellowship AP2005-0544 of the Spanish Ministry of Education, the Project TEC2008-02148 of the Spanish National Research and Development Program, and the Swedish ELLIIT Program.

a vector $u_1u_0u_2$ is:

$$P = 2^2 \cdot u_1 + 2^1 \cdot u_0 + 2^0 \cdot u_2 \tag{2}$$

since $x_2 = u_1$, $x_1 = u_0$ and $x_0 = u_2$. Throughout this paper the notation $P \equiv u_1u_0u_2$ will be used, where the symbol ($\equiv$) relates the decimal and binary representations of a number.

The position can change via bit-dimension permutations. For instance, the permutation $\sigma(u_2u_1u_0) = u_0u_1u_2$ changes dimensions x_2 and x_0. This means that data initially in position $P_0 \equiv u_2u_1u_0$ move to position $P_1 \equiv u_0u_1u_2$.

Finally, algorithms operate on indexed data. The index is represented by $I \equiv b_{n-1}b_{n-2}\dots b_0$. The framework allows the index and position of the data to be known simultaneously, which creates the link between the algorithms and the architectures. For example, if $P \equiv b_0b_2b_1$, the number of dimensions is $n = 3$. Thus, for $I = 5 \equiv 101$, $b_2 = 1$, $b_1 = 0$ and $b_0 = 1$, so its positions is $P(5) \equiv b_0b_2b_1 = 110 \equiv 6$.

III. The Bit Reversal

The bit reversal of $N = 2^n$ indexed data is an algorithm that reorders the data according to a reversing of the bits of the index [1]. This means that any sample with index $I = b_{n-1}\dots b_1b_0$ moves to the place $\mathrm{BR}(I) = b_0b_1\dots b_{n-1}$. Note that the bit reversal is an inversion operation, i.e., $\mathrm{BR}(x) = \mathrm{BR}^{-1}(x)$. Therefore, if data are in natural order, the bit reversal algorithm obtains them in bit-reversed order and viceversa. For instance, the bit reversal of $(0, 1, 2, 3, 4, 5, 6, 7)$ is $(0, 4, 2, 6, 1, 5, 3, 7)$ and the bit reversal of the latter set is the former.

For a hardware circuit that receives a series of N data in natural order, the bit reversal of the data is calculated by the permutation:

$$\sigma(u_{n-1}, \dots, u_1, u_0) = u_0, u_1, \dots, u_{n-1} \tag{3}$$

Due to the inversion property of bit reversal, the same permutation applies to sorting out a sequence in bit-reversed order, as desired for the output frequencies of the FFT.

IV. Circuits for Bit-Dimension Permutation of Serial Data

Any permutation can be broken down into a series of elementary bit-exchanges [21], which are bit-dimension permutations that only interchange two dimensions. In this section, circuits for performing elementary bit-exchanges on serial data are derived using the framework presented in Section II. These circuits are desired to be very simple in order to reduce the hardware complexity. The proposed circuits can be combined in order to carry out any bit-dimension permutation on serial data and are applied to the design of low-complexity circuits for bit reversal in the following sections.

Let us consider $N = 2^n$ data in initial position $P_0 \equiv u_{n-1}, u_{n-2}, \dots, u_0$, i.e., $x_i = u_i$, $\forall i$. According to this, an elementary bit-exchange of two dimensions x_j and x_k, $j > k$, consists of moving each sample in the input position P_0 to the output position P_1 where:

$$\begin{aligned} P_0 &\equiv u_{n-1}, \dots, u_{j+1}, u_j, u_{j-1}, \dots, u_{k+1}, u_k, u_{k-1}, \dots, u_0 \\ P_1 &\equiv u_{n-1}, \dots, u_{j+1}, u_k, u_{j-1}, \dots, u_{k+1}, u_j, u_{k-1}, \dots, u_0 \end{aligned} \tag{4}$$

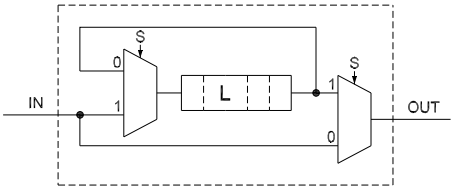


Fig. 1. Basic circuit for the permutation of serial dimensions.

Therefore, samples for which $x_j = x_k$ will remain in the same position, since they fulfill $P_1 = P_0$. Conversely, if $x_j \neq x_k$, the input position corresponds to one of these options:

$$\begin{aligned} P_{0A} &\equiv u_{n-1}, \dots, u_{j+1}, 0, u_{j-1}, \dots, u_{k+1}, 1, u_{k-1}, \dots, u_0 \\ P_{0B} &\equiv u_{n-1}, \dots, u_{j+1}, 1, u_{j-1}, \dots, u_{k+1}, 0, u_{k-1}, \dots, u_0 \end{aligned} \tag{5}$$

If the initial position is $P_0 = P_{0A}$, the elementary bit-exchange moves the sample to $P_1 = P_{0B}$. On the other hand, if $P_0 = P_{0B}$ the output position will be $P_1 = P_{0A}$. Therefore, pairs of samples whose position only differ in dimensions x_j and x_k must be swapped. As a result, an elementary bit-exchange changes the position of half of the N samples, i.e., those for which $x_j \neq x_k$. The rest of the samples are unaffected by the permutation and keep their positions.

In a hardware circuit where samples arrive serially, the position can be defined so that it is equal to the order of arrival [14]. Thus, each sample in position P_0 arrives on time $t = P_0$, where $P_0 = 0$ is the position of the first sample and $P_0 = 2^n - 1$ is the position of the last one. According to this, pairs of input samples that must be interchanged are separated a number of clock cycles equal to:

$$\Delta t = P_{0B} - P_{0A} = 2^j - 2^k \tag{6}$$

Although many pairs of samples have to be permuted, it is important to realize that Δt is constant for all pairs of data. Note that P_{0A} and P_{0B} only differ in dimensions x_j and x_k, leading to the constant value in (6). Consequently, the permutation of serial dimensions consists of delaying some samples and moving other samples forward, always the same number of clock cycles. Furthermore, $L = \Delta t$ registers are necessary for carrying out the delay. Indeed, this is the minimum number of delays that makes the circuit causal and, thus, implementable.

The circuit that carries out this kind of permutation is shown in Fig. 1. It consists of a buffer of length $L = \Delta t$ and two multiplexers controlled by the same control signal, S. The control signal and the length of the buffer depend on the serial dimensions that are permuted. In a general case, if data arrive in series and σ is an elementary bit-exchange of dimensions x_j and x_k, i.e., $\sigma : x_j \leftrightarrow x_k$, the total number of delays of the circuit is:

$$D(\sigma) = L = 2^j - 2^k \tag{7}$$

and the control signal is obtained as:

$$S = \overline{x_j} \ \mathrm{OR} \ x_k \tag{8}$$

Note that $S = 0$ only if $x_j = 1$ and $x_k = 0$, i.e., when a sample in position P_{0B} is at the input of the circuit. At the same time, the sample $P_{0A} = P_{0B} - \Delta t$, which arrived Δt cycles before, is at the output of the buffer. As $S = 0$ both

TABLE I
TIMING DIAGRAM OF THE PERMUTATION $\sigma(u_2u_1u_0) = u_2u_0u_1$.

Time (cycles)	Input Index Dec.	Input Index Bin.	Buffer ($L=1$)	Control (S)	Output Index Dec.	Output Index Bin.
0	0	000	-	1	-	-
1	1	001	0	1	0	000
2	2	010	1	0	2	010
3	3	011	1	1	1	001
4	4	100	3	1	3	011
5	5	101	4	1	4	100
6	6	110	5	0	6	110
7	7	111	5	1	5	101
8	-	-	7	1	7	111

samples are interchanged. Otherwise, $S = 1$ and data are not permuted.

For example, the permutation $\sigma(u_2u_1u_0) = u_2u_0u_1$ interchanges dimensions x_1 and x_0. According to (7) and (8), the circuit in Fig. 1 requires $L = 1$ register in order to carry out the permutation and the control signal is $S = \overline{x_1}$ OR x_0. Table I shows the timing diagram of the circuit. The input sequence arrives in natural order, i.e., $P_0 \equiv b_2b_1b_0$, and the position of the outputs is $P_1 \equiv b_2b_0b_1$. When $S = 1$ samples pass through the buffer. On the other hand, when $S = 0$ the input sample passes directly to the output of the circuit and the output of the buffer is fed back to the input. From the timing diagram it can be observed that outputs are provided one clock cycle after the arrival of the inputs, so the latency is equal to L. The binary representation of the input and output indexes is included in Table I. This allows us to confirm that samples whose indexes differ in bits b_1 and b_0 have been permuted.

Finally, although the description of the circuits has been done in terms of registers or delay elements, it is important to realize that a buffer can be implemented in hardware in many different ways. One option is to use FIFOs. Besides, a buffer can be implemented by a memory whose reading address is delayed with respect to the writing address. In each particular case the most suitable option can be chosen depending on the technology used and the length of the buffer.

V. OPTIMUM CIRCUITS FOR BIT REVERSAL

As has been said, any permutation can be broken down into a series of elementary bit-exchanges. For the case of bit reversal, the permutation σ in equation (3) can be carried out by exchanging each pair of dimensions x_i and x_{n-1-i}.

On the one hand, if the length of the sequence is an even power of 2, then $i \in [0, n/2-1]$, leading to:

$$\sigma = \sigma_{n/2-1} \circ \ldots \circ \sigma_1 \circ \sigma_0 \tag{9}$$

where ($\circ$) represents a function composition and σ_i interchanges dimensions x_i and x_{n-1-i}, i.e,

$$\begin{aligned}\sigma_i(u_{n-1}, \ldots, u_{n-i}, u_{n-1-i}, \ldots, u_{i+1}, u_i, \ldots, u_0) = \\ = u_{n-1}, \ldots, u_{n-i}, u_i, \ldots, u_{i+1}, u_{n-1-i}, \ldots, u_0\end{aligned} \tag{10}$$

which can be also represented as:

$$\sigma_i : x_i \leftrightarrow x_{n-1-i} \tag{11}$$

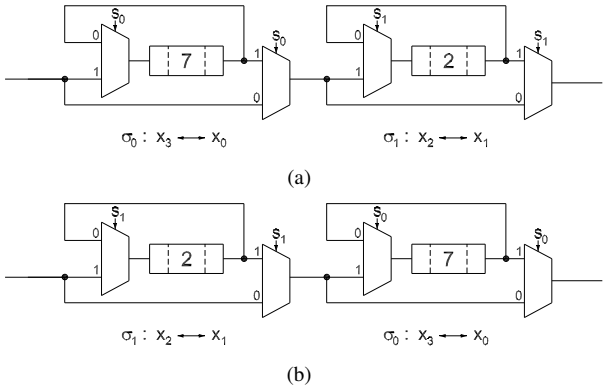
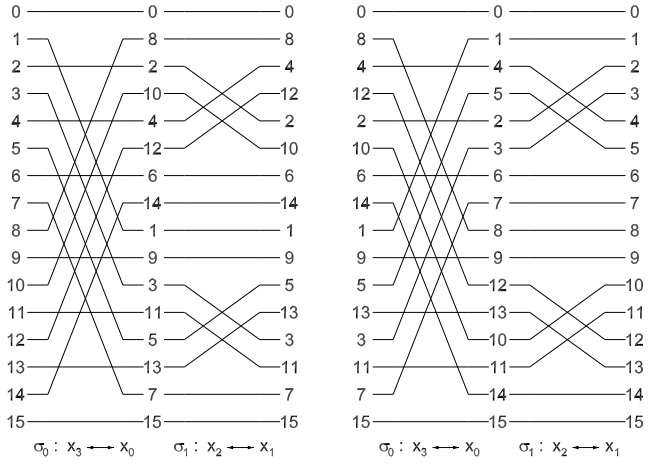


Fig. 2. Circuits for the bit reversal of a 16-point sequence.



(a) Inputs in natural order (b) Inputs in bit-reversed order

Fig. 3. Data shuffling for the bit reversal of a 16-point sequence.

It can be noted that the order in which the permutations σ_i are applied can be arbitrarily chosen, since each of them operates on dimensions different to those of the other permutations. For instance, σ can also be calculated as:

$$\sigma = \sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_{n/2-1} \tag{12}$$

As each permutation σ_i interchanges dimensions x_i and x_{n-1-i}, the number of registers of σ_i is:

$$D(\sigma_i) = 2^{n-1-i} - 2^i \tag{13}$$

and, thus, the total number of registers is:

$$D(\sigma) = \sum_{i=0}^{n/2-1} D(\sigma_i) = 2^n - (2^{n/2+1} - 1) = (\sqrt{N} - 1)^2 \tag{14}$$

The circuit that calculates the bit reversal of a 16-point sequence is shown in Fig. 2(a). It carries out the permutation $\sigma(u_3u_2u_1u_0) = u_0u_1u_2u_3$ in two stages that perform the elementary bit-exchanges $\sigma_0 : x_3 \leftrightarrow x_0$ and $\sigma_1 : x_2 \leftrightarrow x_1$, respectively. For each stage the control signals are obtained from equation (8) and the number of registers is calculated according to equation (7) or (13), being:

$$\begin{aligned}D(\sigma_0) &= 2^3 - 2^0 = 7 \\ D(\sigma_1) &= 2^2 - 2^1 = 2\end{aligned} \tag{15}$$

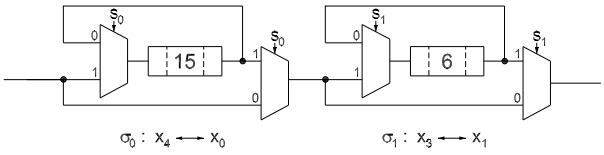


Fig. 4. Circuit for the bit reversal of a 32-point sequence.

Figure 3 shows how the circuit in Fig. 2(a) shuffles the samples. Figure 3(a) represents the case of inputs arriving in natural order, whereas in Fig. 3(b) the shuffling is applied to sort out an input sequence in bit-reversed order. It can be observed that σ_0 and σ_1 only exchange samples whose positions differ in 7 and 2, respectively, as happens in the circuit in Fig. 2(a). Besides, as the order of the elementary bit-exchanges can be arbitrarily chosen, σ_1 can also be calculated before σ_0. This solution is shown in Fig. 2(b).

On the other hand, if the length of the sequence is an odd power of 2 the problem is slightly different. In this case the number of dimensions is odd and the intermediate one is not permuted. According to this, the bit reversal is calculated by permuting each pair of dimensions x_i and x_{n-1-i} for $i \in [0,(n-3)/2]$, leading to:

$$\sigma = \sigma_{(n-3)/2} \circ \ldots \circ \sigma_1 \circ \sigma_0 \tag{16}$$

In this case, the number of registers of each permutation is also calculated according to (13) and the total number is:

$$D(\sigma) = \sum_{i=0}^{(n-3)/2} D(\sigma_i) = \left(\sqrt{2N}-1\right)\left(\sqrt{\frac{N}{2}}-1\right) \tag{17}$$

The circuit that calculates the bit reversal of a 32-point sequence is shown in Fig. 4. It carries out the permutation $\sigma(u_4u_3u_2u_1u_0) = u_0u_1u_2u_3u_4$ by means of the elementary bit-exchanges $\sigma_0 : x_4 \leftrightarrow x_0$ and $\sigma_1 : x_3 \leftrightarrow x_1$. According to this, the lengths of the buffers are:

$$\begin{aligned} D(\sigma_0) &= 2^4 - 2^0 = 15 \\ D(\sigma_1) &= 2^3 - 2^1 = 6 \end{aligned} \tag{18}$$

The analysis of the circuit is analogous to the previous cases.

Finally, in the proposed circuits the total size of the buffers is lower than the length of the sequence, $N = 2^n$. This is, indeed, the lowest number of registers that are necessary to carry out the bit reversal. This can be demonstated as follows. If samples arrive in natural order and are provided in bit reversal, each sample with index I will be the I-th sample that arrives at the circuit and the BR(I)-th sample that leaves the circuit. Therefore, the circuit must allocate at least $I - \mathrm{BR}(I)$ samples. The lowest necessary number of registers is given by the value that maximizes this subtraction, i.e., $D_{\min} = \max(I - \mathrm{BR}(I))$. This equation is maximum when the half most significant bits of I are equal to "1" and the rest of the bits are equal to "0", leading to the same values for $D_{\min}$ as those in (14) and (17). The case of samples arriving in bit-reversed order and provided in natural order is analogous.

VI. Extension to Other Radices

The order of the output frequencies of the FFT depends on the radix used in the decomposition. For radix-2 and any radix-$2^k, \forall k$, samples arrive in bit-reversed order, so the circuits proposed in the previous section can be used. However, for radices such as radix-4 and radix-8 the outputs are provided in a different order. These cases are analyzed in this section.

The bit reversal for radix-4 (BR_4) performs a bit reversal by considering the bits of the index in pairs, i.e., each sample with index I moves to $\mathrm{BR}_4(I)$, where:

$$\begin{aligned} I &\equiv b_{n-1}b_{n-2}, b_{n-3}b_{n-4}, \ldots, b_3b_2, b_1b_0 \\ \mathrm{BR}_4(I) &\equiv b_1b_0, b_3b_2, \ldots, b_{n-3}b_{n-4}, b_{n-1}b_{n-2} \end{aligned} \tag{19}$$

The commas in equation (19) have been added in order to highlight the pairs of bits. Notice also that $N = 2^n$ must be a power of four and, therefore, n must be an even number.

As $\mathrm{BR}_4(I)$ is an inversion operation, a circuit that permutes:

$$\begin{aligned} &\sigma(u_{n-1}, u_{n-2}, u_{n-3}, u_{n-4}, \ldots, u_3, u_2, u_1, u_0) = \\ &= u_1, u_0, u_3, u_2, \ldots, u_{n-3}, u_{n-4}, u_{n-1}, u_{n-2}, \end{aligned} \tag{20}$$

can be used to calculate the BR_4 of a sequence in natural order and viceversa. This permutation can be carried out by a series of elementary bit-exchanges, σ_i, where:

$$\begin{aligned} \sigma_{2i} &: x_{2i} \leftrightarrow x_{n-2i-2} \\ \sigma_{2i+1} &: x_{2i+1} \leftrightarrow x_{n-2i-1} \end{aligned} \tag{21}$$

leading to the following registers for each permutation:

$$\begin{aligned} D(\sigma_{2i}) &= 2^{n-2i-2} - 2^{2i} \\ D(\sigma_{2i+1}) &= 2^{n-2i-1} - 2^{2i+1} \end{aligned} \tag{22}$$

If the number of samples is an even power of four, $i \in [0, n/4-1]$, and the total number of registers is:

$$D(\sigma) = \sum_{i=0}^{n/4-1} [D(\sigma_{2i}) + D(\sigma_{2i+1})] = (\sqrt{N}-1)^2 \tag{23}$$

If the number of samples is an odd power of four, $i \in [0,(n-6)/4]$ and the total number of registers is:

$$D(\sigma) = \sum_{i=0}^{(n-6)/4} [D(\sigma_{2i}) + D(\sigma_{2i+1})] = (\sqrt{4N}-1)(\sqrt{\frac{N}{4}}-1) \tag{24}$$

The bit reversal for radix-8 (BR_8) can be analyzed in the same way. In this case the elementary bit-exchanges are:

$$\begin{aligned} \sigma_{3i} &: x_{3i} \leftrightarrow x_{n-3i-3} \\ \sigma_{3i+1} &: x_{3i+1} \leftrightarrow x_{n-3i-2} \\ \sigma_{3i+2} &: x_{3i+2} \leftrightarrow x_{n-3i-1} \end{aligned} \tag{25}$$

For even powers of eight the total number of registers is $(\sqrt{N}-1)^2$, whereas for odd powers of eight:

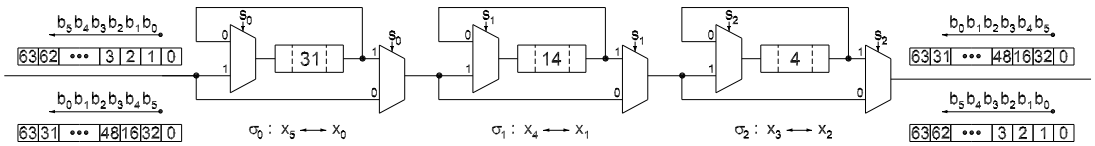
$$D(\sigma) = \left(\sqrt{8N}-1\right)\left(\sqrt{\frac{N}{8}}-1\right) \tag{26}$$

Generalizing these results, the algorithm for any radix-r (BR_r) groups $\log_2 r$ bits of the index and carries out a reversal of these groups. Besides, BR_r is an inversion for any r. If N is an even power of r, the total number of registers is:

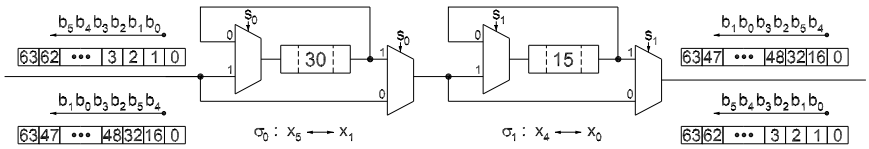
$$D(\sigma) = \left(\sqrt{N}-1\right)^2 \tag{27}$$

whereas for odd powers of r the total number of registers is:

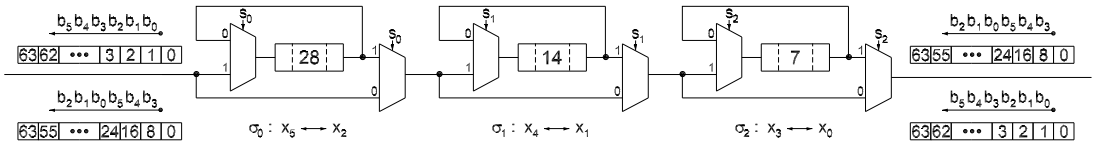
$$D(\sigma) = \left(\sqrt{rN}-1\right)\left(\sqrt{\frac{N}{r}}-1\right) \tag{28}$$



(a) Radix-2



(b) Radix-4



(c) Radix-8

Fig. 5. Bit reversal circuits of a 64-point sequence for radices 2, 4 and 8.

In all cases the latency is equal to the number of registers.

Figure 5 shows bit reversal circuits of a 64-point sequence for radices 2, 4 and 8. Input sequences both in natural and bit-reversed order are shown. As 64 is an even power of 2 and 8, the total number of registers for radices 2 and 8 is calculated according to equation (27), leading to 49 in both cases. Conversely, 64 is an odd power of 4. According to equation (28) the total number of registers for radix-4 is 45.

Finally, in previous approaches the bit reversal is calculated using either double buffering [12], [15] or a single memory of N addresses [12], [14], [16]. Thus, the radix-2 and radix-8 circuits in Fig. 5 save 15 storing elements (23%), whereas the radix-4 one saves 19 (29%). For higher N the savings are larger, although they represents a lower percentage of storing elements. For instance, 31 (12%) registers are saved for 256 points and radix-2, and 71 (13%) for 512 points and radix-8.

VII. Conclusions

This paper presents new circuits for calculating the bit reversal of a sequence of data. The circuits are optimum in two senses. On the one hand, they have the lowest possible latency. On the other hand, they require the lowest number of storage elements. This number is lower than the length of the sequence and, therefore, lower than the number of storage elements used in previous memory-based approaches.

The circuits are very suitable for sorting out the output frequencies of the FFT in pipelined architectures. As the output order depends on the radix of the FFT, different radices have been analyzed and optimum circuits for any general case have been presented.

References

[1] B. Gold and C. M. Rader, *Digital Processing of Signals*. New York: McGraw Hill, 1969.

[2] D. Sundararajan, M. O. Ahmad, and M. Swamy, "A fast FFT bit-reversal algorithm," *IEEE Trans. Circuits Syst. II*, vol. 41, no. 10, Oct. 1994.

[3] J. Rius and R. D. Porrata-Dòria, "New FFT bit-reversal algorithm," *IEEE Trans. Signal Process.*, vol. 43, no. 4, pp. 991–994, Apr. 1995.

[4] J. Prado, "A new fast bit-reversal permutation algorithm based on a symmetry," *IEEE Signal Process. Lett.*, vol. 11, no. 12, Dec. 2004.

[5] M. Jaber and D. Massicotte, "A novel approach for FFT data reordering," in *Proc. IEEE Int. Conf. Symp. Circuits Syst.*, May 2010, pp. 1615–1618.

[6] Z. Zhang and X. Zhang, "Cache-optimal methods for bit-reversals," in *Proc. ACM/IEEE Conf. Supercomputing*, 1999.

[7] K. Gatlin and L. Carter, "Memory hierarchy considerations for fast transpose and bit-reversals," in *Proc. Int. Symp. High-Performance Computer Arch.*, 1999, pp. 33–42.

[8] C.-M. Chen, C.-C. Hung, and Y.-H. Huang, "An energy-efficient partial FFT processor for the OFDMA communication system," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 2, pp. 136–140, Feb. 2010.

[9] T. Choinski and T. Tylaska, "Generation of digit reversed address sequences for fast Fourier transforms," *IEEE Trans. Comput.*, vol. 40, no. 6, pp. 780–784, Jun. 1991.

[10] S. H. Ok and B. I. Moon, "A digit reversal circuit for the variable-length radix-4 FFT," in *Proc. Future Generation Comm. Networking*, vol. 2, Dec. 2007, pp. 496–500.

[11] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 7, pp. 585–589, Jul. 2006.

[12] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.

[13] M. Sánchez, M. Garrido, M. López, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.

[14] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Universidad Politécnica de Madrid, 2009.

[15] F. Kristensen, P. Nilsson, and A. Olsson, "Flexible baseband transmitter for OFDM," in *Proc. IASTED Conf. Circuits Signals Syst.*, May 2003.

[16] T. Chakraborty and S. Chakrabarti, "On output reorder buffer design of bit reversed pipelined continuous data FFT architecture," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2008, pp. 1132–1135.

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

[18] T. Järvinen, "Systematic methods for designing stride permutation interconnections," Ph.D. dissertation, Tampere Univ. of Technology, 2004.

[19] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1991.

[20] D. Fraser, "Array permutation by index-digit permutation," *J. Assoc. Comp. Machinery (ACM)*, vol. 23, no. 2, pp. 298–309, Apr. 1976.

[21] A. Edelman, S. Heller, and L. Johnsson, "Index transformation algorithms in a linear algebra framework," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 12, pp. 1302–1309, Dec. 1994.