Chapter 3
Classification of FFT Processor Algorithms

The computational complexity of the Discrete Fourier transform (DFT) is very high. It requires $(N\text{-}1)^2$ complex multiplications and $N(N\text{-}1)$ complex additions [5]. As an alternative to direct computation of the DFT, we can use Fast Fourier Transform (FFT) algorithms to efficiently calculate the DFT. Many computationally efficient FFT algorithms have been proposed. This chapter presents a brief collection of the FFT algorithms and their properties.

3.1 Discrete Fourier Transform (DFT)

Given a length N sequence x(n), a DFT generated complex sequence X(k) can be defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \quad k = 0,1,\ldots,N-1$$
$$where \quad Twiddle\ Factor, W_N^{nk} = e^{-j\frac{2\pi}{N}nk} \tag{3.1}$$

If the periodic and symmetric properties of the twiddle factor W_N^m are exploited, then the computation of X[k] will be more efficient.

$$symmetric : W_N^{m+N/2} = -W_N^m$$
$$periodic : W_N^{m+N} = W_N^m$$

Two types of FFT algorithms are *Decimation in Time* (DIT) and *Decimation in Frequency* (DIF). The former is based on decomposition of the input sequence into smaller subsequences and the latter is based on dividing the output sequence into smaller subsequences [5]. The computational complexity of these two types are the

same. In this chapter, we only introduce the DIF FFT algorithms.

3.2 Fixed Radix Algorithms

3.2.1 Radix-2 DIF FFT Algorithm

The radix-2 algorithm [5] is the most common fixed algorithm. This algorithm can only compute the data having a power of 2 length. The radix-2 butterfly calculation is very simple among the overall class of FFT algorithms. Each butterfly stage reads two data inputs and produces two outputs. The definition of an *N*-point DFT is given again as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \quad k = 0, 1, ..., N-1$$

The input sequence, *x*(n) can be separated into the first half of the sequence and the second half. The mathematical expressions are defined in the following equations:

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{nk} + \sum_{n=0}^{N/2-1} x(n+\frac{N}{2}) W_N^{(n+N/2)k} \\ &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{nk} + W_N^{(N/2)k} \sum_{n=0}^{N/2-1} x(n+\frac{N}{2}) W_N^{nk} \end{aligned} \tag{3.2}$$

where,

$$W_N^{(N/2)k} = e^{-j\frac{2\pi}{N}\frac{N}{2}k} = \cos(\pi k) - j\sin(\pi k) = (-1)^k$$

We get

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} x(n) W_N^{nk} + (-1)^k \sum_{n=0}^{N/2-1} x(n+\frac{N}{2}) W_N^{nk} \\ &= \sum_{n=0}^{(N/2)-1} [x(n) + (-1)^k x(n+\frac{N}{2})] W_N^{nk} \end{aligned} \tag{3.3}$$

The even output represented in Equation (3.2) is obtained by the summation of the first half and the second half of the input sequence for each of the *N*/2 point DFTs.

The odd output represented in Equation (3.3) is obtained by the subtraction of the first half and the second half of the input sequence for each of the $N/2$ point DFTs and multiplying this subtractive result by W_N^n. For the first stage, 2r and 2r+1 is substituted for index k in Equation (3.3), we get the Equation (3.4) and (3.5).

$$\begin{aligned} X[2r] &= \sum_{n=0}^{(N/2)-1} (x[n] + x[n+(N/2)]) W_{N/2}^{rn} \\ &= \sum_{n=0}^{(N/2)-1} g[n] W_{N/2}^{rn} \end{aligned} \tag{3.4}$$

$$\begin{aligned} X[2r+1] &= \sum_{n=0}^{(N/2)-1} (x[n] - x[n+(N/2)]) W_N^n W_{N/2}^{rn} \\ &= \sum_{n=0}^{(N/2)-1} h[n] W_N^n W_{N/2}^{rn} \end{aligned} \tag{3.5}$$

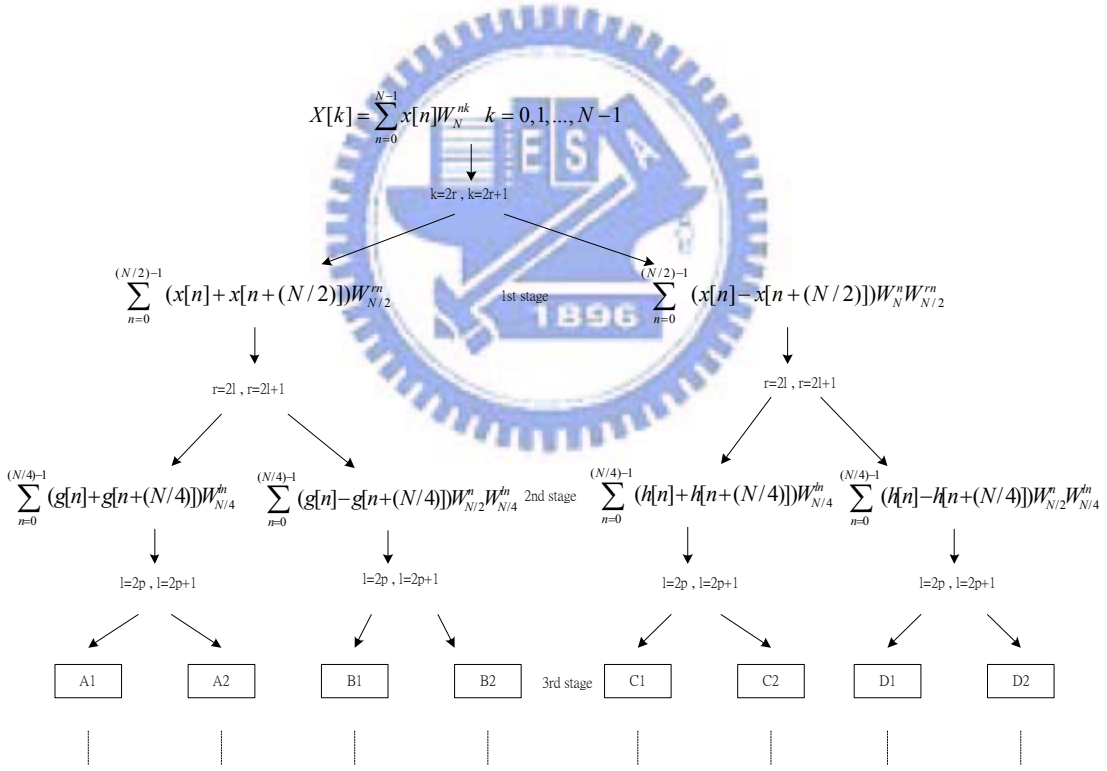


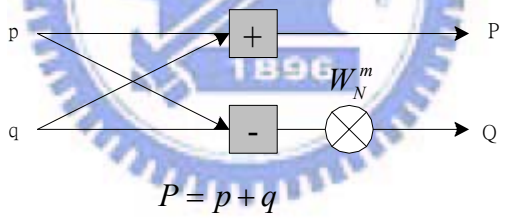
Figure 3.1 Separation procedure for radix-2 algorithm

Figure 3.1 shows the basic concept of the separation procedure for the radix-2 algorithm. This procedure can be applied recursively until $\log_2 N$ stages are produced. That is, the final stage has been reduced to 2-point DFT computations.

Among the stages, there is a *N*/2 butterfly structure in each stage, as shown in Figure 3.2. Radix-2 butterfly element requires only two complex adders and one complex multiplier. For this reason, its hardware cost is very low.

The FFT operation has *N* inputs and *N* outputs, so we need *N* registers to store all the output data from all the butterfly stage calculations. If we use in-place computation, it will not be necessary to use N storage registers. To reduce registers, we reuse the same registers to store successive iterations of the results of each butterfly. This kind of calculation is called an in-place computation [5].

The procedure shown in Figure 3.1 is depicted with a complete *N*=16-point example in Figure 3.3. It is obvious that when the input sequences of the DIF FFT algorithm are arranged in normal order, its output sequences are arranged in bit-reversed order after using the in-place computation method.



$$P = p + q$$

$$Q = (p - q)W_N^m$$

Figure 3.2 Butterfly structure of radix-2 DIF FFT

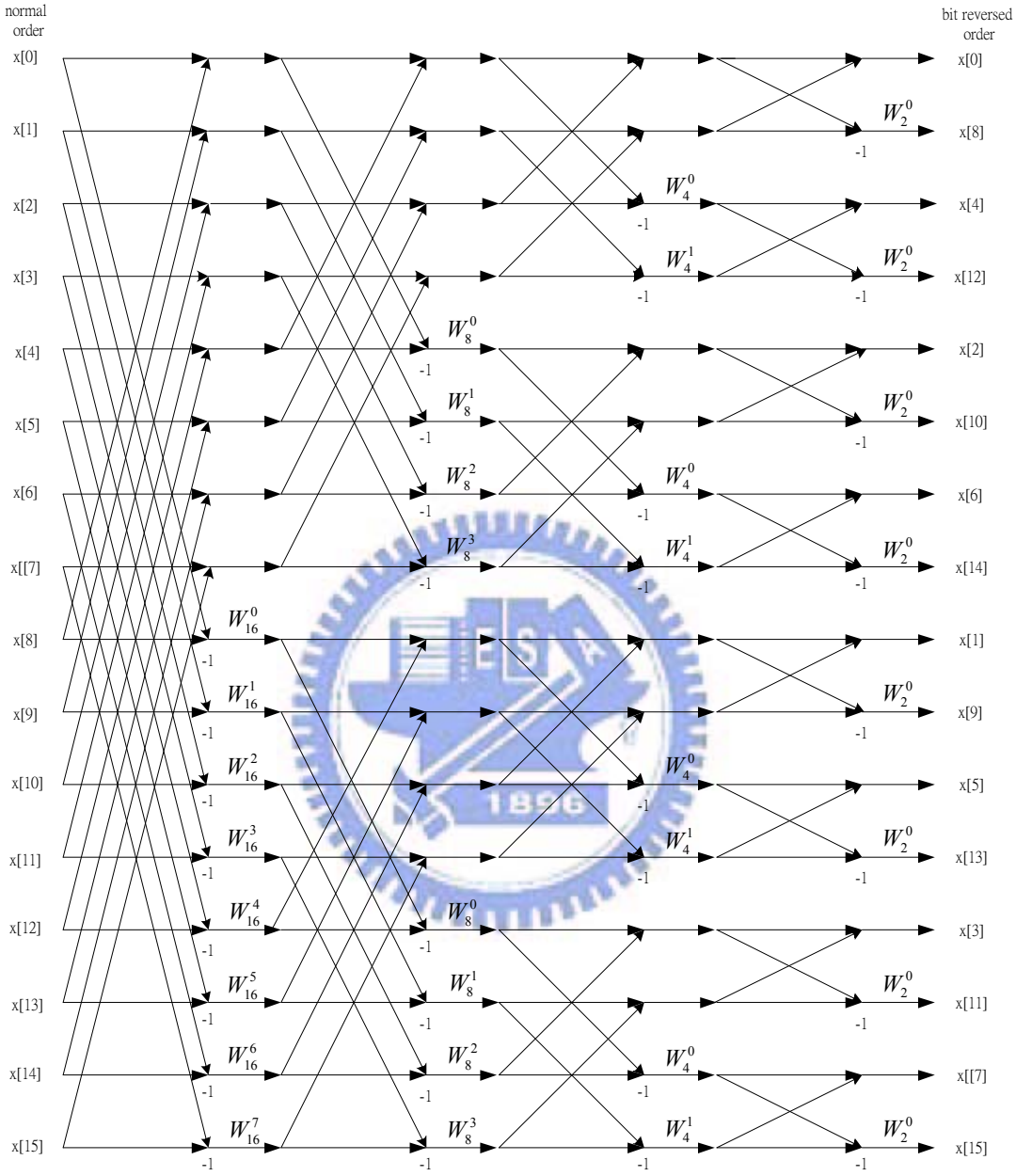


Figure 3.3 Signal flow graph of complete decomposition of a 16 point radix-2 DIF FFT

3.2.2 Radix-4 DIF FFT Algorithm

Each radix-4 butterfly stage reads four data inputs and produces four data outputs. For this reason, this algorithm provides higher computation speed compared to the radix-2 algorithm. Decomposing Equation (3.4) and (3.5) into four groups of frequency components: 4u, 4u+1, 4u+2, 4u+3, the radix 4 algorithm [6] can be obtained.

$$X[4u] = \sum_{n=0}^{(N/4)-1} \{(x[n] + x[n+(N/2)]) + (x[n+N/4] + x[n+(3N/4)])\} W_{N/4}^{nu} \quad (3.6)$$

$$X[4u+1] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+(N/2)]) - j(x[n+N/4] - x[n+(3N/4)])\} W_N^n W_{N/4}^{nu} \quad (3.7)$$

$$X[4u+2] = \sum_{n=0}^{(N/4)-1} \{(x[n] + x[n+(N/2)]) - (x[n+N/4] + x[n+(3N/4)])\} W_N^{2n} W_{N/4}^{nu} \quad (3.8)$$

$$X[4u+3] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+(N/2)]) + j(x[n+N/4] - x[n+(3N/4)])\} W_N^{3n} W_{N/4}^{nu} \quad (3.9)$$

The butterfly signal flow graph of Equation (3.6) to Equation (3.9) is depicted in Figure 3.4, where the interval n is 0~(N/4)-1. We can observe that the radix-4 butterfly needs only the three complex multipliers W_N^n, W_N^{2n}, W_N^{3n} and eight complex adders for each stage. Multiplication by the $-j$ term causes the real and imaginary parts of the data to be exchanged. The signal flow graph in Figure 3.5 is illustrated for the computation of a16-point DFT using the radix-4 algorithm.

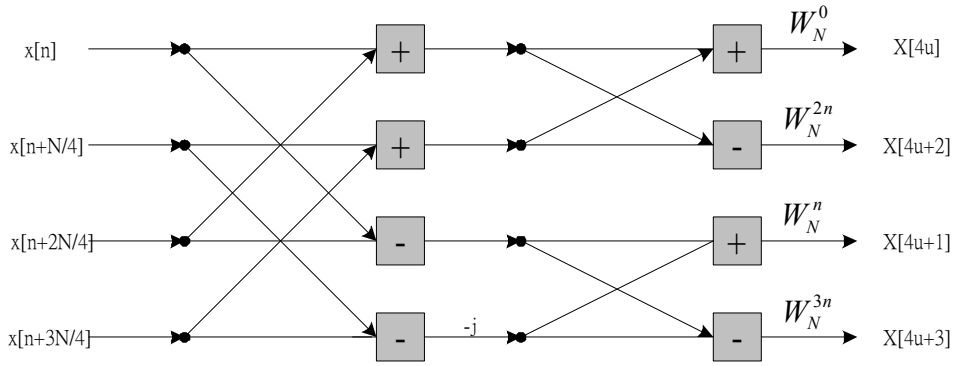


Figure 3.4 Butterfly structure for radix-4 algorithm

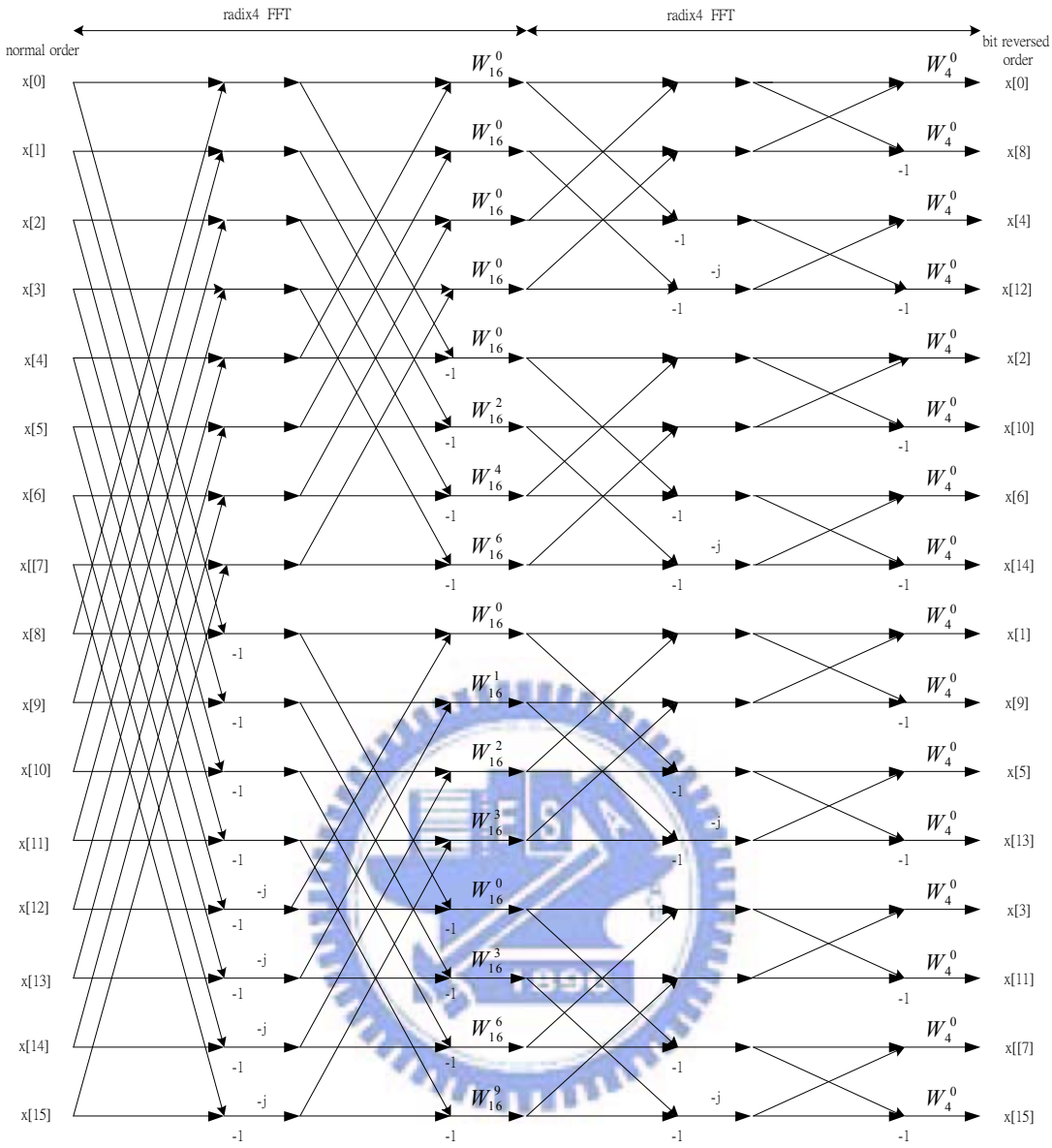


Figure 3.5 Signal flow graph of complete decomposition of a 16 point radix-4 DIF FFT

3.2.3 Radix-8 DIF FFT Algorithm

Based on the two algorithms presented in the previous subsection, we conclude that with increasing the radix, the cost in hardware will increase linearly, complexity will increase, and implementation difficulty will increase. However, when using higher radix algorithms (e.g. radix8), the number of complex multiplications

decreases [17]. In this case, the common direct mapping method of radix-r butterfly element operation is more complicated and has larger area. So, the implementation of the butterfly element structure is of great interest to decrease the hardware complexity for higher radix algorithm. The duty of a designer is how to arrange and design the data flow of the butterfly element structures to achieve small area and high performance. Here, we introduce radix-8 FFT algorithm.

Each radix-8 butterfly stage reads eight data inputs and produces eight data outputs. Radix-8 algorithm [8] is expressed in Equation (3.10) through (3.17).

$$X[8k+0]=\sum_{n=0}^{N/8-1}[\{(x[n]+x[n+N/2])+(x[n+N/4]+x[n+3N/4])\}+ \{(x[n+N/8]+x[n+5N/8])+(x[n+3N/8]+x[n+7N/8])\}]W^{8nk} \tag{3.10}$$

$$X[8k+1]=\sum_{n=0}^{N/8-1}[\{(x[n]-x[n+N/2])-j(x[n+N/4]-x[n+3N/4])\}+W^{N/8}. \{(x[n+N/8]-x[n+5N/8])-j(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{n} \tag{3.11}$$

$$X[8k+2]=\sum_{n=0}^{N/8-1}[\{(x[n]+x[n+N/2])-(x[n+N/4]+x[n+3N/4])\}-j \{(x[n+N/8]-x[n+5N/8])+j(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{2n} \tag{3.12}$$

$$X[8k+3]=\sum_{n=0}^{N/8-1}[\{(x[n]-x[n+N/2])+j(x[n+N/4]-x[n+3N/4])\}+W^{3N/8}. \{(x[n+N/8]-x[n+5N/8])+j(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{3n} \tag{3.13}$$

$$X[8k+4]=\sum_{n=0}^{N/8-1}[\{(x[n]+x[n+N/2])+(x[n+N/4]+x[n+3N/4])\}- \{(x[n+N/8]-x[n+5N/8])+(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{4n} \tag{3.14}$$

$$X[8k+5]=\sum_{n=0}^{N/8-1}[\{(x[n]-x[n+N/2])-j(x[n+N/4]-x[n+3N/4])\}-W^{N/8}. \{(x[n+N/8]-x[n+5N/8])-j(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{5n} \tag{3.15}$$

$$X[8k+6]=\sum_{n=0}^{N/8-1}[\{(x[n]+x[n+N/2])-(x[n+N/4]-x[n+3N/4])\}+j \{(x[n+N/8]+x[n+5N/8])-(x[n+3N/8]+x[n+7N/8])\}]W^{8nk}W^{6n} \tag{3.16}$$

$$X[8k+7]=\sum_{n=0}^{N/8-1}[\{(x[n]-x[n+N/2])+j(x[n+N/4]-x[n+3N/4])\}-W^{3N/8}. \{(x[n+N/8]-x[n+5N/8])+j(x[n+3N/8]-x[n+7N/8])\}]W^{8nk}W^{7n} \tag{3.17}$$

The above eight equations can be depicted with the butterfly structure in Figure 3.6. This structure fully utilizes the twiddle factor with its properties of symmetry and

periodicity. It requires two constant multipliers $W_N^{N/8}$ *and* $W_N^{3N/8}$, seven non-trivial complex multipliers and three trivial factors *–j*. The two constant multipliers with $W_N^{N/8}$ *and* $W_N^{3N/8}$ can be replaced by 12 additions by using shift adders [9]. The computation of a 8-point DFT accomplished by the radix-8 algorithm is depicted in Figure 3.7.

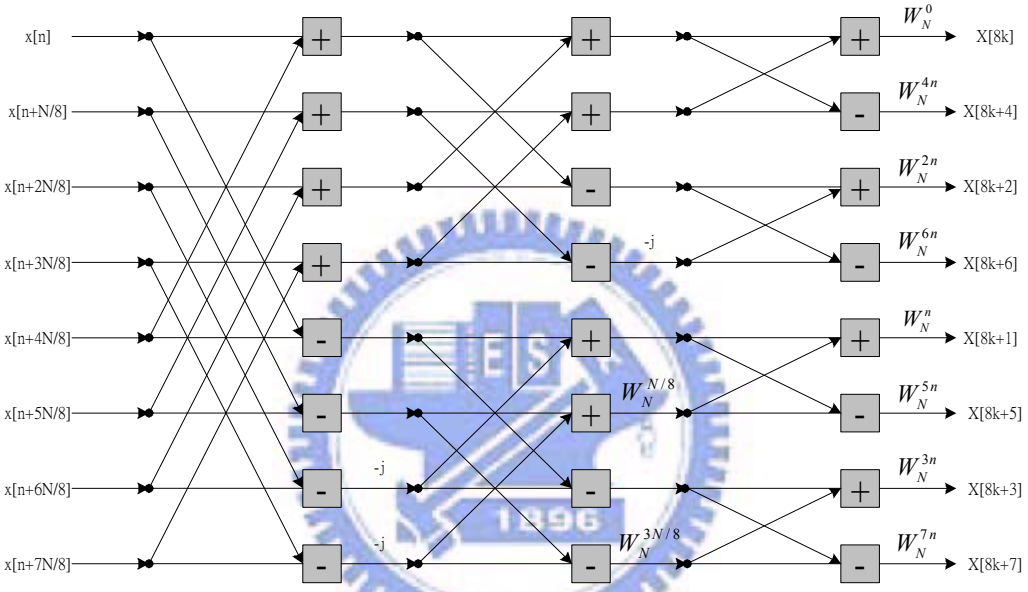


Figure 3.6 Butterfly structure of radix-8 algorithm

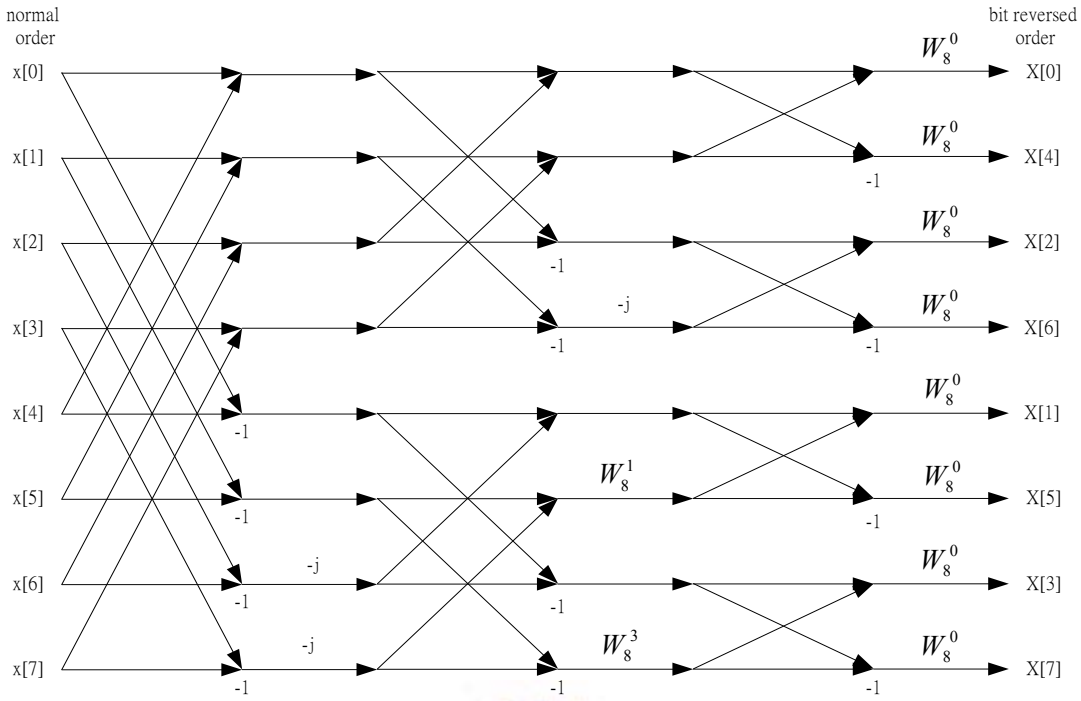


Figure 3.7 Signal flow graph of a 8-point radix-8 DIF FFT

3.3 Split Radix Algorithms

References [7] and [10] present the fundamental idea of the split radix algorithm. The split radix algorithm specifies that an *N*-point DFT can be implemented by decomposing different parts of the input sequence using different algorithms. Split radix algorithms such as the radix-2/4 split algorithm, and the radix-2/8 split algorithm have yet to be presented. Although they require less complex multipliers, they are not often used in IC design due to their irregular structure.

3.3.1 Split Radix-2/4 Algorithm

This algorithm [10] utilizes a radix-2 algorithm to calculate the even-indexed output sequences and a radix-4 algorithm to calculate the odd-indexed output

sequences. This idea can be represented by the mathematical expression in Equation (3.18) through (3.20).

For the even-indexed term,

$$X[2u] = \sum_{n=0}^{(N/2)-1} (x[n] + x[n+(N/2)])W_{N/2}^{un} . \quad (3.18)$$

For the odd-indexed term,

$$X[4u+1] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+(N/2)]) - j(x[n+N/4] - x[n+(3N/4)])\}W_N^n W_{N/4}^{nu} \quad (3.19)$$

$$X[4u+3] = \sum_{n=0}^{(N/4)-1} \{(x[n] - x[n+(N/2)]) + j(x[n+N/4] - x[n+(3N/4)])\}W_N^{3n} W_{N/4}^{nu} . \quad (3.20)$$

Equation (3.19) and (3.20) can be represented by the following butterfly element structure including twiddle factor coefficients. Figure 3.9 is the computation of a 16-point split radix-2/4 algorithm.

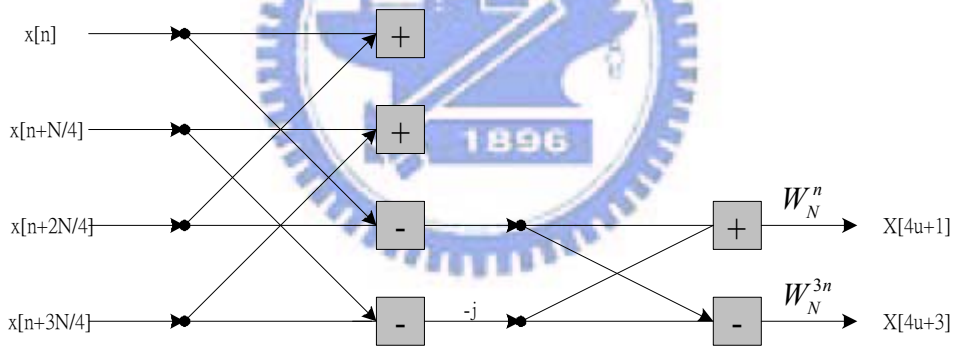


Figure 3.8 Butterfly structure of split radix-2/4 algorithm

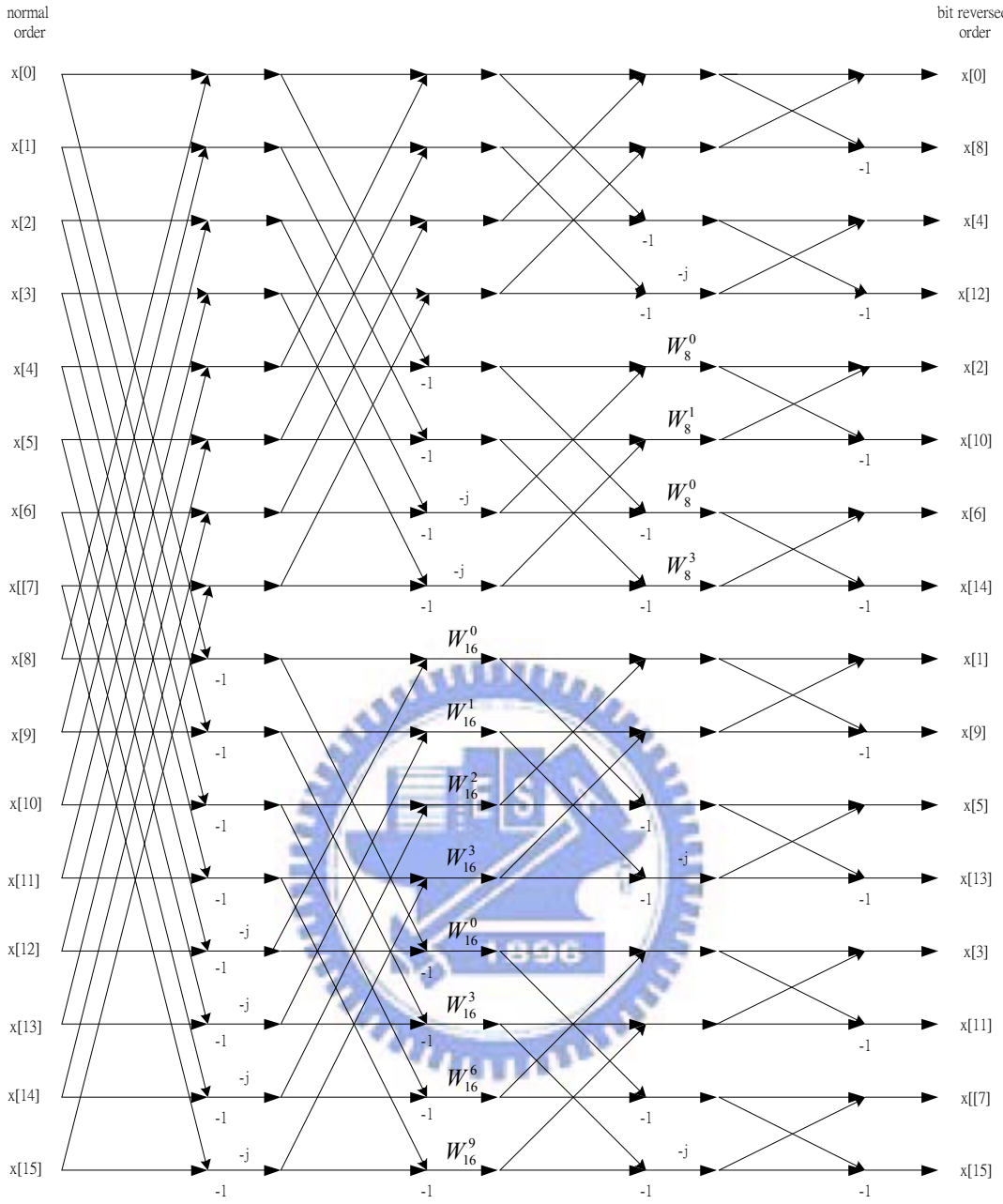


Figure 3.9 Computation of a 16-point split radix-2/4 algorithm

3.3.2 Split Radix-2/8 Algorithm

The split radix-2/8 algorithm [7] [9] utilizes a radix-8 algorithm to calculate the odd-indexed output instead of a radix-4 algorithm. The even-indexed output is calculated the same way as in the radix-2/4 algorithm. That is, the radix-2/8 algorithm

is based on the decomposition of the length-N DFT sequence into one length-$N/2$ DFT and four the length-$N/8$ DFTs.

For the even-indexed term with one length-$N/2$ DFTs,

$$X[2u] = \sum_{n=0}^{(N/2)-1} (x[n] + x[n+(N/2)])W_{N/2}^{un} . \tag{3.21}$$

For the odd-indexed term with four length-$N/8$ DFTs,

$$X[8u+1] = \sum_{n=0}^{(N/8)-1} \{[(x[n]-x[n+(N/2)]) - j(x[n+N/4]-x[n+3N/4])]+W_N^{N/8} \times[(x[n+N/8]-x[n+5N/8]) - j(x[n+3N/8]-x[n+7N/8])]\}W_N^{n}W_{N/8}^{nu} \tag{3.22}$$

$$X[8u+3] = \sum_{n=0}^{(N/8)-1} \{[(x[n]-x[n+(N/2)]) + j(x[n+N/4]-x[n+3N/4])]+W_N^{3N/8} \times[(x[n+N/8]-x[n+5N/8]) + j(x[n+3N/8]-x[n+7N/8])]\}W_N^{3n}W_{N/8}^{nu} \tag{3.23}$$

$$X[8u+5] = \sum_{n=0}^{(N/8)-1} \{[(x[n]-x[n+(N/2)]) - j(x[n+N/4]-x[n+3N/4])]-W_N^{N/8} \times[(x[n+N/8]-x[n+5N/8]) - j(x[n+3N/8]-x[n+7N/8])]\}W_N^{5n}W_{N/8}^{nu} \tag{3.24}$$

$$X[8u+7] = \sum_{n=0}^{(N/8)-1} \{[(x[n]-x[n+(N/2)]) + j(x[n+N/4]-x[n+3N/4])]-W_N^{3N/8} \times[(x[n+N/8]-x[n+5N/8]) + j(x[n+3N/8]-x[n+7N/8])]\}W_N^{7n}W_{N/8}^{nu} \tag{3.25}$$

To make the idea of the radix-2/8 algorithm easier to understand, the simplified butterfly structure is illustrated in Figure 3.10. A 16-point split radix-2/8 algorithm is shown in Figure 3.11.

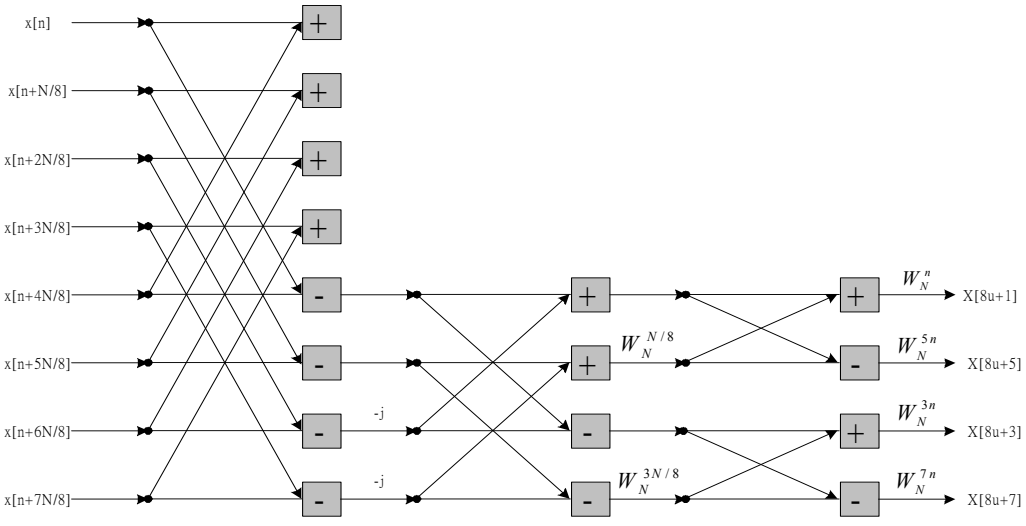


Figure 3.10 Butterfly structure of radix-2/8 algorithm

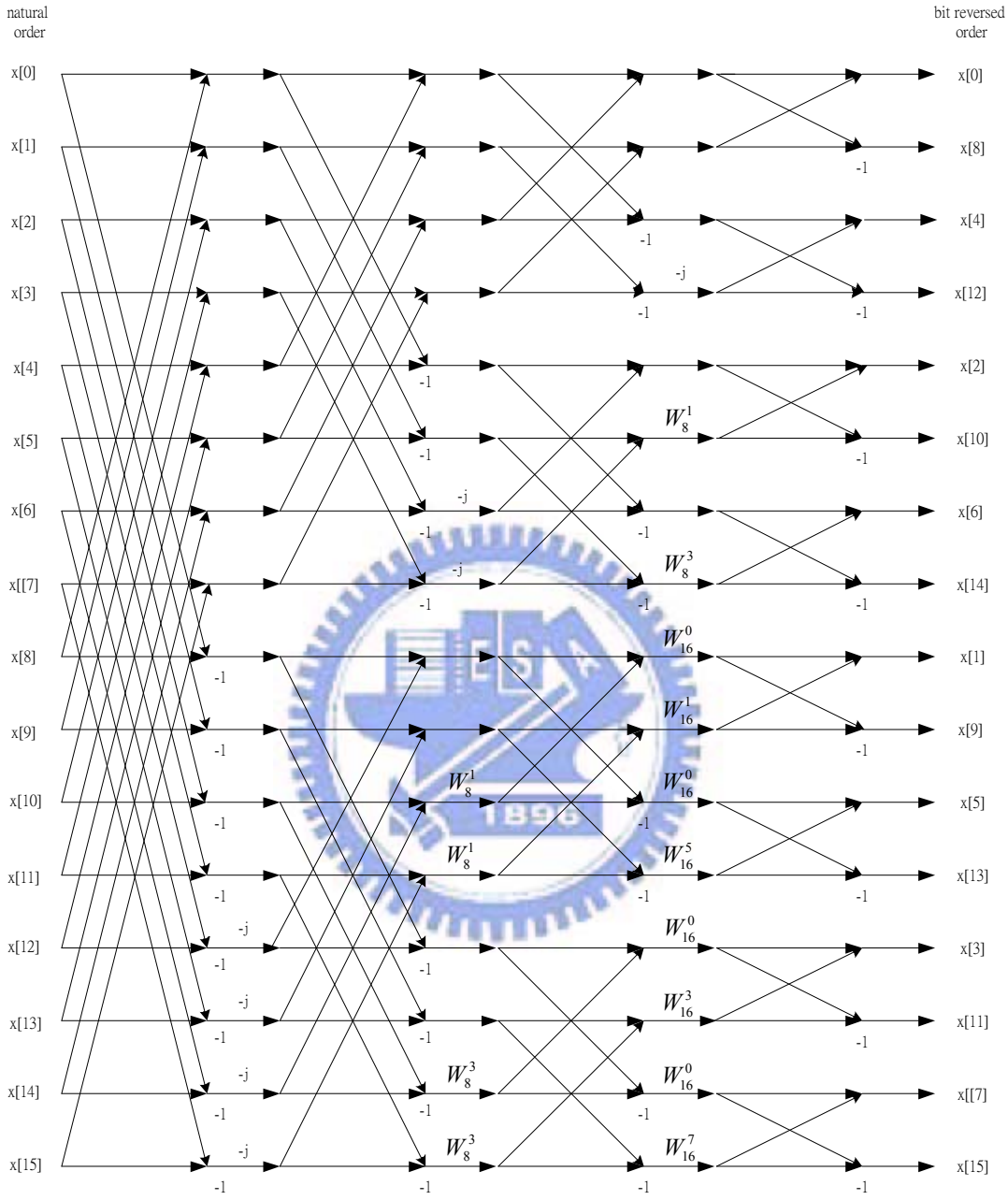


Figure 3.11 A 16-point split radix-2/8 algorithm

3.4 Mixed Radix Algorithm

The radices of the butterfly structure in different FFT computation stages need not be equal. Such an algorithm is called a *mixed radix algorithm* [11]. When the FFT size is not a power of radix-*r*, we can only use a mixed radix algorithm to design an FFT. For example, a 1024-point sequence is not a power of 8. Thus, we can combine three stages with a radix-8 algorithm and one stage with a radix-2 algorithm to operate the transform over a 1024-point data sequence.