DFT—THE DISCRETE FOURIER TRANSFORM

The *discrete Fourier transform* (*DFT*) is defined as

$$X(k) \overset{\Delta}{=} \sum_{n=0}^{N-1} x(n) W^{nk} \qquad , \ k = 0, 1, \ldots, N-1$$

and the *inverse discrete Fourier transform* (*IDFT*) is

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W^{-nk} \qquad , \ n = 0, 1, \ldots, N-1$$

where $W = e^{-j2\pi/N}$

Direct Computation of the DFT

```
Program Direct_DFT;
var
    x, Y: array[0..Nminus1] of Complex;
begin
    for k := 0 to N-1 do
        begin
            Y[k] := x[0];
            for n := 1 to N-1 do
            Y[k] := Y[k] + Wnk * x[n];
        end;
end.
```

FFT — THE FAST FOURIER TRANSFORM ALGORITHM

In 1965, Cooley and Tukey developed a fast algorithm based on the divide-and-conquer principle which requires only $O(N \log_2(N))$ complex operations.

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

The Cooley-Tukey FFT

Radix-2 FFT

```
Program CT_FFT;
   const
       N = 1024;  M = 10;  Nminus1 = 1023; { N = 2^M }
   type
       Complex = record
           re : Double; im : Double;
       end;
       C_array : array[0..Nminus1] of Complex;
   var
       Stage, Ns, M1, k, kNs,  p, q : integer;
       WCos, WSin, TwoPiN, TempRe, TempIm : Double;
       x : C_array;

   begin  { READ INPUT DATA INTO x }
       Ns := N;  M1 := M;
       TwoPiN := 2 * Pi/N;
```

```
        for Stage := 1 to M do
            begin
                k := 0;
                Ns := Ns div 2;
                M1 := M1 – 1;
                while k < N do
                begin
                for q := 1 to Ns  do
                begin
                p := Digit_Reverse(k/2^M1);
                WCos := cos(TwoPiN * p); {W to the power of p }
                WSin := –sin(TwoPiN * p); {W = exp(–j2π/N) }
                kNs := k + Ns;
                TempRe := x[kNs].re * WCos – x[kNs].im * WSin;
                TempIm := x[kNs].im * WCos + x[kNs].re * WSin;
                x[kNs].re := x[k].re – TempRe;
                x[kNs].im := x[k].im – TempIm;
                x[k].re := x[k].re + TempRe;
                x[k].im := x[k].im + TempIm;
                k := k + 1;
                end;
                k := k + Ns;
                end;
            end;
            Unscramble; { OUTPUT DATA STORED IN x }
    end.
```
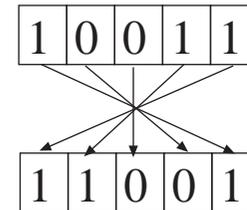
Digit Reversal

```
function Digit_Reverse(Digit: Integer) : Integer;
var
    N, q, NewAddr, Rmbit, OldAddr : Integer;
begin
    NewAddr := 0;
    OldAddr := Digit;
    for q := 1 to M do
        begin
        Rmbit := OldAddr mod 2;
        OldAddr := OldAddr div 2;
        if Rmbit = 1 then
            NewAddr := NewAddr * 2 + 1
        else
            NewAddr := NewAddr + NewAddr;
        end;
    Digit_Reverse := NewAddr;
end;
```

1	0	0	1	1

1	1	0	0	1

DSP Integrated Circuits
Lars Wanhammar
Department of Electrical Engineering
Linköping University
larsw@isy.liu.se
http://www.es.isy.liu.se/

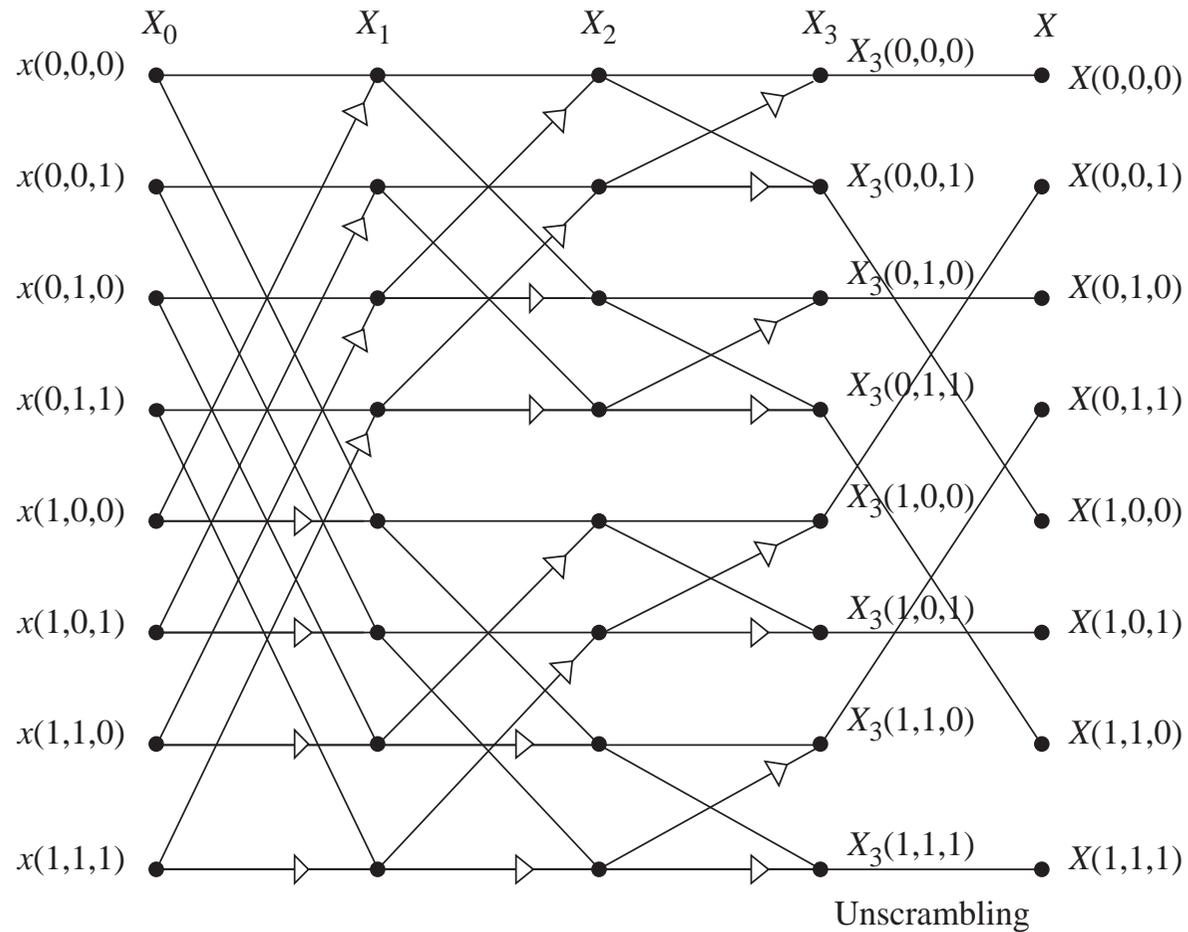# Unscramble

```
procedure Unscramble;
    var
        temp : Complex;
        k, q : integer;
    begin
        for k := 0 to N - 1 do
            begin
            q := Digit_Reverse(k);
            if q > k then
                begin
                temp := x[k];
                x[k] := x[q];
                x[q] := temp;
                end;
            end;
    end;
```
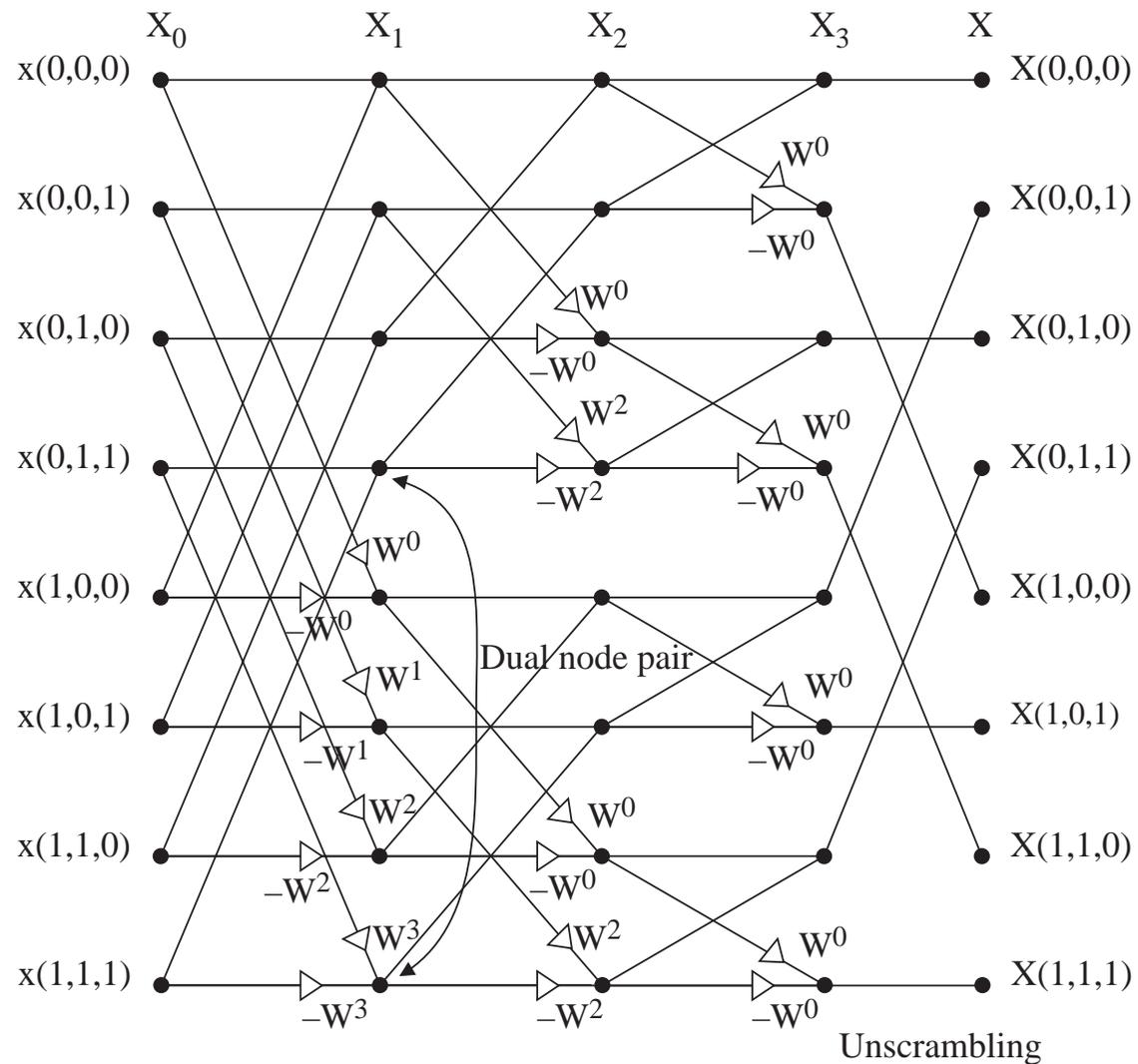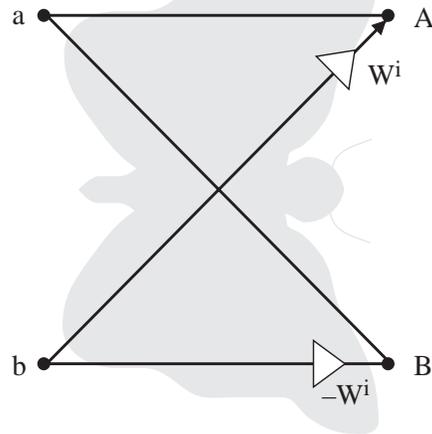
DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

Cooley-Tukey FFT Signal-Flow Graph

Unscrambling

Sande-Tukey's FFT Signal-Flow Graph

Radix-2 Butterfly

Butterfly op

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/
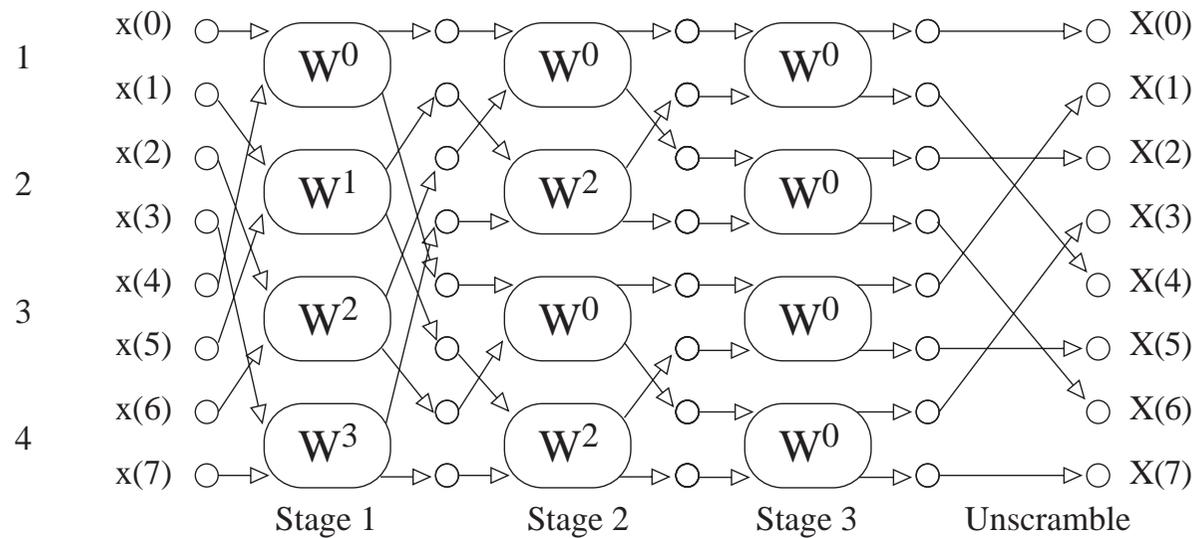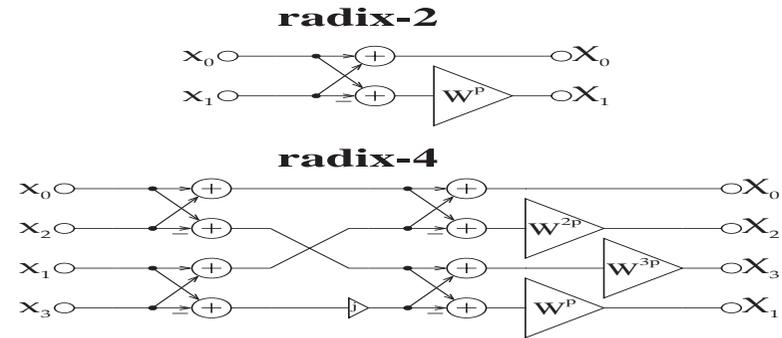
Radix-4 Butterfly

Radix-8 Butterfly
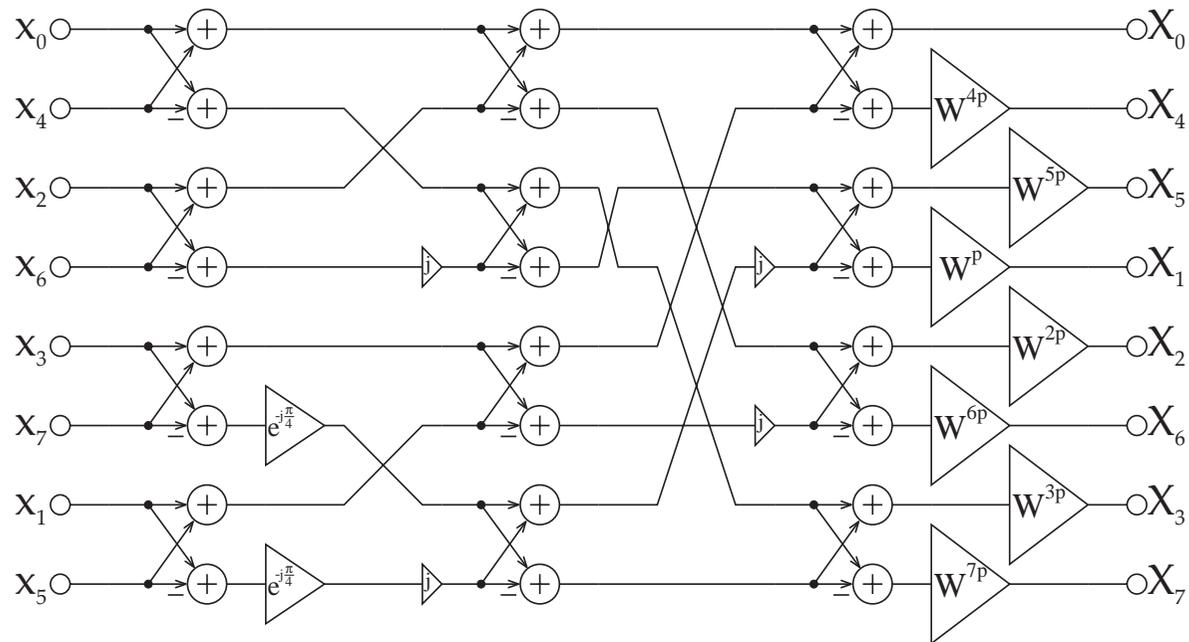
Table 1: Radix-*r* 4096-points FFT Algorithms

Operation	Radix-2	Radix-4	Radix-8
Complex Multiplications	22528	15360	10752
Real Multiplications	0	0	8192
Complex Additions	49152	49152	49152
Real Additions	0	0	8192
Memory Accesses	49152	24576	16384

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

The Inverse FFT

The inverse FFT (*IFFT*) can be computed by a simple modification of the input read process.

The IFFT is obtained by computing the FFT of the following sequence

$$x(n) = \begin{cases} \dfrac{X(0)}{N} & \text{for } n = 0 \\[2ex] \dfrac{X(N-n)}{N} & \text{for } n = 1, 2, \ldots, N-1 \end{cases}$$

Thus, the first value is stored at address 0 while the rest are stored in reverse order. This operation can easily be implemented in hardware by changing the address lines to the memory or by using an up–down counter that is properly initiated.

Another method to compute the IFFT is to first interchange the real and imaginary parts, then perform the FFT, and, finally, interchange the real and imaginary parts. We will use this method in an implementation of an FFT processor.

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

FFT PROCESSOR—CASE STUDY 1

Develop an FFT processor usable VLSI building block.

In order to be flexible so that the processor, the performance in terms of computational throughput, word length, and transform length should be easily modifiable.

Specification

The processor shall compute a 1024-point complex FFTs with a continuous throughput of more than 2000 FFTs per second.

The processor shall also be able to perform both the FFT and the IFFT.

The host is a general purpose 32-bit computer.

We assume that the data rate is a modest 16 MHz.

The data word length for both input and output is selected to be 16 bits for the real and imaginary parts.

The internal data word length is, in this early stage of the design process, estimated to 21 bits.

The required coefficient word length is assumed to be14 bits

We arbitrarily select to implement the Sande–Tukey FFT.

System Design Phase

The first step in the system design phase is to partition the computation of an FFT into three consecutive processes;

- reading the input data from the host,
- performing the FFT/, and
- writing the result into the memory of the host.

The requirement is that the execution time for the FFT, including I/O, should not exceed 0.5 ms.

The I/O transfer frequency should be only 16 MHz. Input and output data to the FFT processor are transferred as two 16-bit real numbers.

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

A total of 1024 complex numbers are transferred to and from the FFT processor.

The time needed for the I/O is

$$t_{I/O} = \frac{2 \cdot 1024}{16 \cdot 10^6} = 0.128 \text{ ms}$$

It is possible to overlap the I/O operations, i.e., writing and reading data, and computation of the FFT, thereby effectively extending the available time for computing the FFT. For the sake of simplicity we assume that this possibility is not exploited.

The available time for the FFT is therefore 0.372 ms.

The I/O processes will handle both the rearranging of data that is required to compute the IFFT and the unscrambling of the data array in the last stage of the FFT.

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/

The IFFT can be obtained by interchanging the real and imaginary parts, when the data are both written into and read from the memory.

The unscrambling can be accomplished by reversing the address lines to the memory when reading data out of the FFT processor.

Hence, both of these tasks can be accomplished without any extra processing time.

In fact, this is an example of merging operations and communications!

DSP Integrated Circuits
Lars Wanhammar

Department of Electrical Engineering
Linköping University

larsw@isy.liu.se
http://www.es.isy.liu.se/