2006-1557: UNFOLDING THE WINGS OF THE BUTTERFLY: AN ALTERNATIVE EXPLANATION FOR FFTS

Kathleen Ossman, University of Cincinnati

Dr. Kathleen Ossman is an assistant professor in the Electrical and Computer Engineering Technology Department at the University of Cincinnati. She received a BSEE and MSEE from Georgia Tech in 1982 and a Ph.D. from the University of Florida in 1986. Her interests include feedback control systems and digital signal processing.

Unfolding the Wings of the Butterfly:
An Alternative Explanation for FFTs

Abstract

This paper discusses an approach to teaching Fast Fourier Transforms (FFTs) to engineering technology students using a set of graphics that not only illustrates how the FFT algorithm works but also gives students an idea of how an FFT algorithm might be programmed.

Introduction

One of the most difficult topics to teach in an introductory course in Digital Signal Processing is Fast Fourier Transforms (FFTs) which are simply efficient algorithms for computing Discrete Fourier Transforms (DFTs) in real-time. Most textbooks[1-6] begin with an explanation of how the data points are divided into specific pairs followed by a set of complicated “subscript heavy” re-combining equations used to calculate larger FFTs from the smaller subset FFTs. These equations are typically followed with a Butterfly diagram designed to help visualize an FFT algorithm. The Butterfly diagram illustrates the progression of the re-ordered data through a series of interwoven multiply and add operations to reach the final FFT. In this paper, a quick summary of the traditional textbook approach to FFTs will be presented followed by a handout developed by the author to make the process easier for students to visualize.

Traditional Approach

Most digital signal processing textbooks[2-6] present some variation of the following set of equations. An FFT algorithm simply breaks a DFT down into several 2-point DFTs by exploiting the symmetry properties of the twiddle factors: $W_N = e^{-j2\pi/N} = \cos(2\pi/N) - j \sin(2\pi/N)$.

$$X_{DFT}(k) = \sum_{n=0}^{N-1} x(n)\,(W_N)^{nk} = \underbrace{\sum_{n=0}^{N/2-1} x(2n)\,(W_N)^{2nk}}_{\text{Even Indices}} + \underbrace{\sum_{n=0}^{N/2-1} x(2n+1)\,(W_N)^{(2n+1)k}}_{\text{Odd Indices}}$$

$$X_{DFT}(k) = \sum_{n=0}^{N/2-1} x(2n)\,(W_{N/2})^{nk} + (W_N)^k \sum_{n=0}^{N/2-1} x(2n+1)\,(W_{N/2})^{nk} = \text{Two } N/2\text{-point DFTs}$$

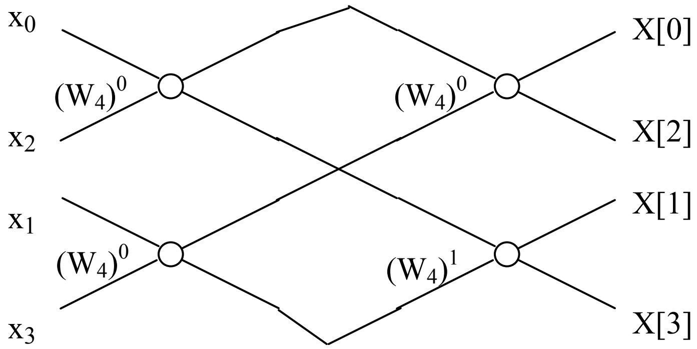
Using the even samples [x(0), x(2), … x(N-2)], compute an N/2-point DFT → $X^{even}(k)$
Using the odd samples [x(1), x(3), … x(N – 1)], compute an N/2-point DFT → $X^{odd}(k)$

For Recombining:

$$\begin{aligned} X_{DFT}(k) &= X^{even}(k) + (W_N)^k X^{odd}(k) \qquad k = 0, 1, \ldots, N/2-1 \\ X_{DFT}(k + N/2) &= X^{even}(k) - (W_N)^k X^{odd}(k) \qquad k = 0, 1, \ldots, N/2-1 \end{aligned}$$

Use same procedure to break each N/2-point DFT in half ⇒ Four N/4-point DFTs.
Continue the procedure until problem has been reduced to a set of 2-point DFTs.

These equations are accompanied by an explanation of how input data is broken down to allow pair by pair or 2-point DFTs which are recombined to form larger DFTs. The most commonly used visual tool is the Butterfly Diagram shown below for a 4-point FFT.



I have used this traditional explanation for several years in my digital signal processing courses for electrical engineering technology students. Students were give an assignment or two which forced them to trace through the butterfly diagram and compute an 8-point DFT by hand. Even with this assignment, it was clear that they were not getting a good grasp of the concept of an FFT as an efficient computing method for DFTs. They were overwhelmed by the complicated mathematical equations and lost in the maze of the butterfly diagram.

Illustrative Handout

In spring 2005, I delivered my traditional lecture of FFTs and once again observed total confusion in the faces of my students. The next lecture period, I distributed a handout which was developed to give the students a more visual explanation of the process. The difference between this lecture and the previous one was amazing: my students were totally engaged, asked lots of questions, and commented very positively on the handout. They were easily able to visualize the extension to a 16-point or higher FFT. I was really pleased when two students had enough confidence in their comprehension of the material to point out a couple of errors in my equation blocks (which by the way were corrected for this paper). Before presenting the handout, a few comments are in order. Although the handout does give students a sense of memory allocation and computational requirements, it is not meant to be a flow chart for programming an FFT. The sole purpose is to give students a visual picture of the break down and recombining process in an FFT along with an understanding of the weighting factors.

FFTs: An Efficient Method for Computing DFTs
(32-ELTN-487)

4-PT DFT

<u>DFT</u> $X_{DFT}(k) = \sum_{n=0}^{3} x(n)e^{-j2\pi nk/4} = \sum_{n=0}^{3} x(n)\,(W_N)^{nk}$ $\quad k = 0,1,2,3$

$$[X_{DFT}(0)\ X_{DFT}(1)\ X_{DFT}(2)\ X_{DFT}(3)] = [x(0)\ x(1)\ x(2)\ x(3)] \begin{pmatrix} (W_4)^0 & (W_4)^0 & (W_4)^0 & (W_4)^0 \\ (W_4)^0 & (W_4)^1 & (W_4)^2 & (W_4)^3 \\ (W_4)^0 & (W_4)^2 & (W_4)^4 & (W_4)^6 \\ (W_4)^0 & (W_4)^3 & (W_4)^6 & (W_4)^9 \end{pmatrix}$$

$$[X_{DFT}(0)\ X_{DFT}(1)\ X_{DFT}(2)\ X_{DFT}(3)] = [x(0)\ x(1)\ x(2)\ x(3)] \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix}$$

<u>Programming a direct 4-Point DFT</u>

Memory Allocation:

- 1 4-element data buffer for input
- 1 4x4 Array of Weights

Processing:

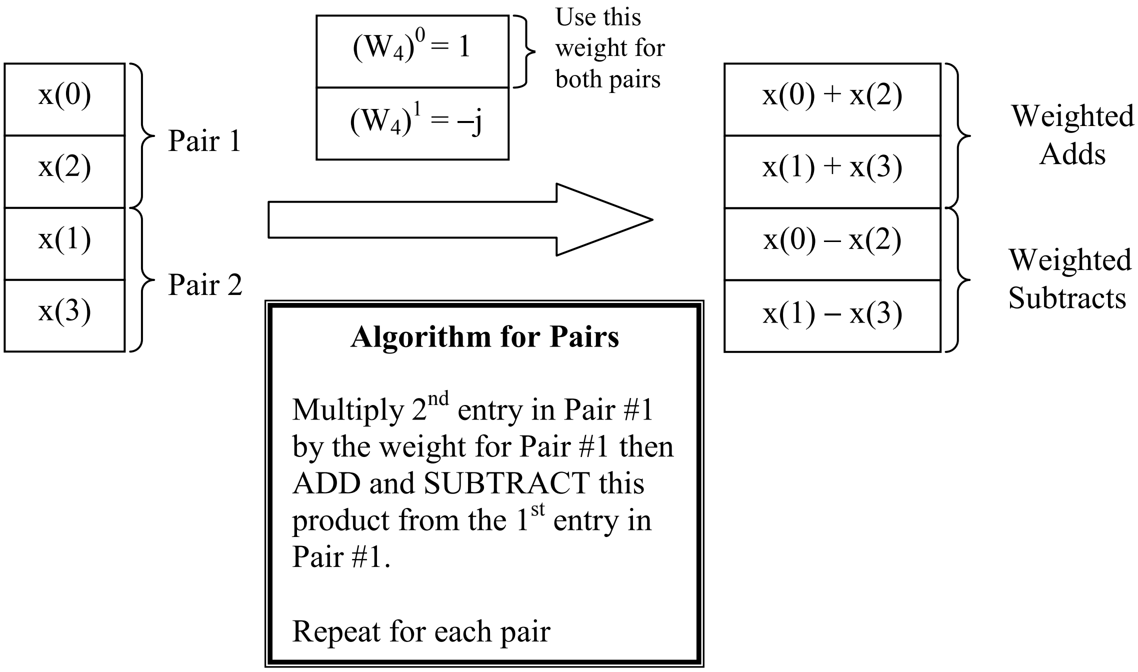
$X_{DFT}(0)$ = Data Buffer times 1st column of Weight Array (4 multiplies)
$X_{DFT}(1)$ = Data Buffer times 2nd column of Weight Array (4 multiplies)
$X_{DFT}(2)$ = Data Buffer times 3rd column of Weight Array (4 multiplies)
$X_{DFT}(3)$ = Data Buffer times 4th column of Weight Array (4 multiplies)

Total Multiplies = N^2 = 16

Comment: Obviously, some of the weights are 1 so with some creative programming the number of multiplies could be reduced.

4-PT FFT

Level 1: 2-pt FFTs



x(0)

x(2)

Pair 1

x(1)

x(3)

Pair 2

$(W_4)^0 = 1$

$(W_4)^1 = -j$

Use this weight for both pairs

x(0) + x(2)

x(1) + x(3)

Weighted Adds

x(0) – x(2)

x(1) – x(3)

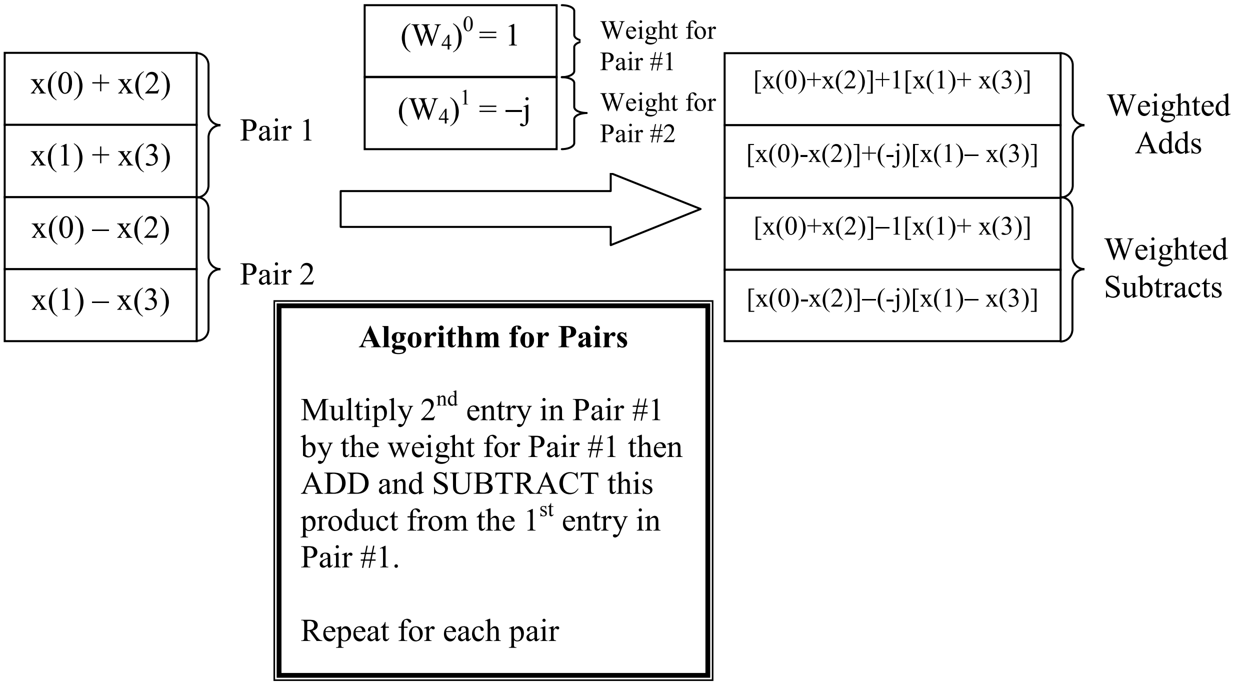
Weighted Subtracts

Algorithm for Pairs

Multiply 2nd entry in Pair #1 by the weight for Pair #1 then ADD and SUBTRACT this product from the 1st entry in Pair #1.

Repeat for each pair

Level 2: 4-pt FFT



x(0) + x(2)

x(1) + x(3)

Pair 1

x(0) – x(2)

x(1) – x(3)

Pair 2

$(W_4)^0 = 1$

Weight for Pair #1

$(W_4)^1 = -j$

Weight for Pair #2

[x(0)+x(2)]+1[x(1)+ x(3)]

[x(0)-x(2)]+(-j)[x(1)– x(3)]

Weighted Adds

[x(0)+x(2)]–1[x(1)+ x(3)]

[x(0)-x(2)]–(-j)[x(1)– x(3)]

Weighted Subtracts

Algorithm for Pairs

Multiply 2nd entry in Pair #1 by the weight for Pair #1 then ADD and SUBTRACT this product from the 1st entry in Pair #1.

Repeat for each pair

Programming a 4-Point FFT

Memory Allocation:

- 1 4-element data buffer for input (re-ordered)
- 1 4-element data buffer for intermediate results
- 1 2-element Array of Weights (Need only half due to 180^o symmetry)

Processing:

Level 1: Each pair (N/2 of them) requires one multiply (2 multiplies)
Level 2: Each pair (N/2 of them) requires one multiply (2 multiplies)

Total Multiplies = (N/2)*Number of Levels = $(N/2)\log_2(N) = 4$

8-PT DFT

DFT $$X_{DFT}(k) = \sum_{n=0}^{7} x(n)e^{-j2\pi nk/8} = \sum_{n=0}^{7} x(n)\,(W_N)^{nk} \qquad k = 0,1, \ldots 7$$

Programming a direct 8-Point DFT

Memory Allocation:

- 1 8-element data buffer for input
- 1 8x8 Array of Weights

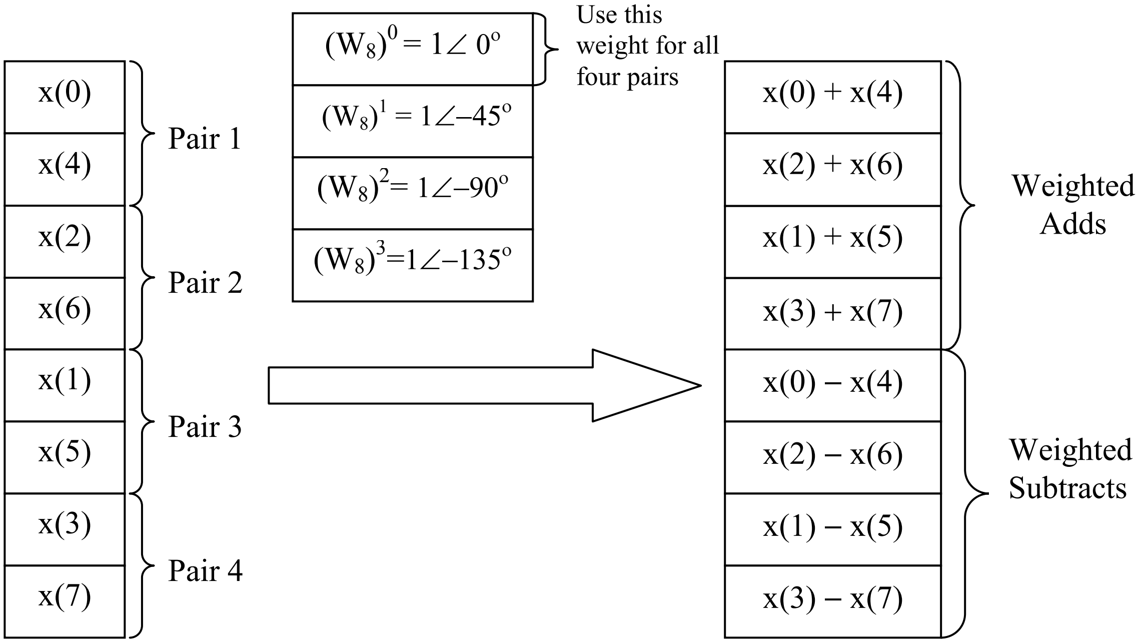
Processing:

$X_{DFT}(0)$ = Data Buffer times 1st column of Weight Array (8 multiplies)
$X_{DFT}(1)$ = Data Buffer times 2nd column of Weight Array (8 multiplies)
$X_{DFT}(2)$ = Data Buffer times 3rd column of Weight Array (8 multiplies)
$X_{DFT}(3)$ = Data Buffer times 4th column of Weight Array (8 multiplies)
$X_{DFT}(4)$ = Data Buffer times 5th column of Weight Array (8 multiplies)
$X_{DFT}(5)$ = Data Buffer times 6th column of Weight Array (8 multiplies)
$X_{DFT}(6)$ = Data Buffer times 7th column of Weight Array (8 multiplies)
$X_{DFT}(7)$ = Data Buffer times 8th column of Weight Array (8 multiplies)

Total Multiplies = $8^2 = 64$

8-PT FFT

Level 1: Four 2-pt FFTs



$(W_8)^0 = 1\angle\, 0^o$
$(W_8)^1 = 1\angle{-45}^o$
$(W_8)^2 = 1\angle{-90}^o$
$(W_8)^3 = 1\angle{-135}^o$

Use this weight for all four pairs

x(0)	Pair 1
x(4)	
x(2)	Pair 2
x(6)	
x(1)	Pair 3
x(5)	
x(3)	Pair 4
x(7)	

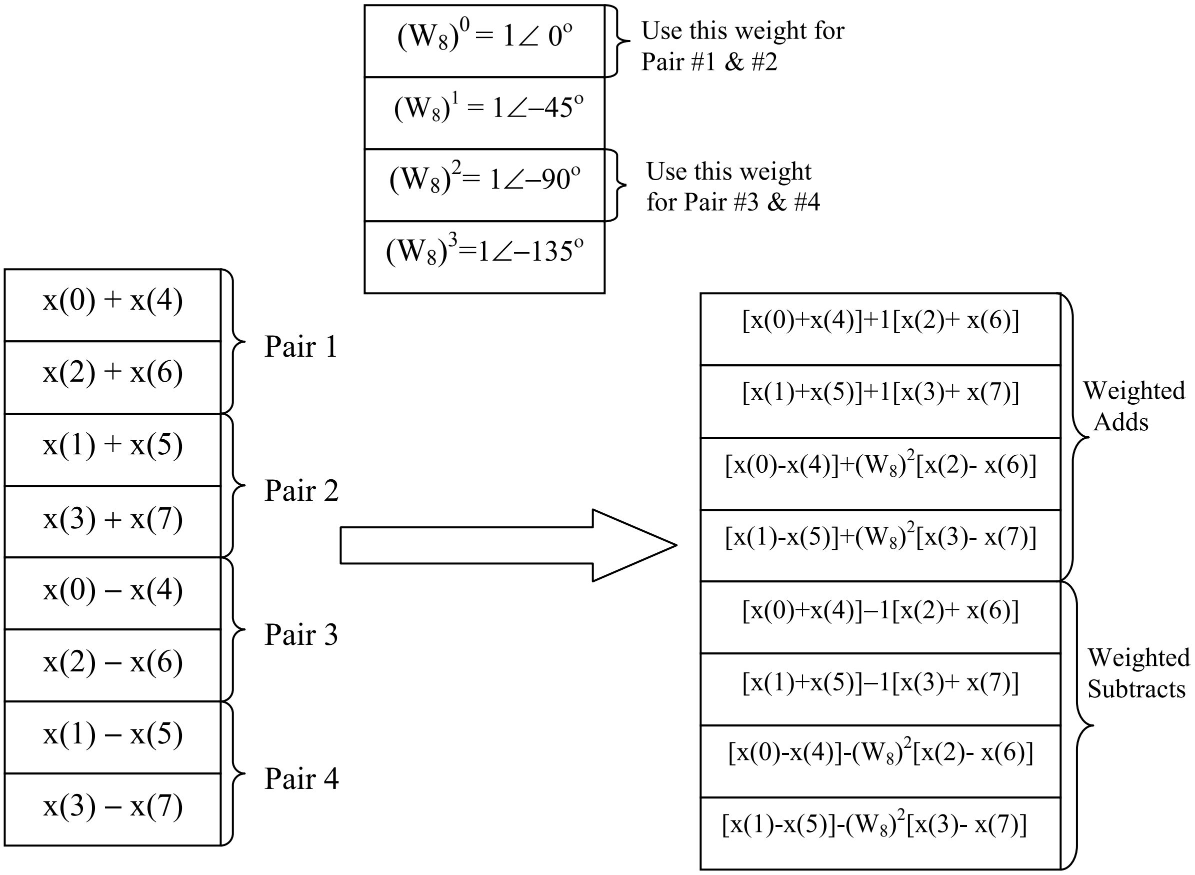
x(0) + x(4)	Weighted Adds
x(2) + x(6)	
x(1) + x(5)	
x(3) + x(7)	
x(0) − x(4)	Weighted Subtracts
x(2) − x(6)	
x(1) − x(5)	
x(3) − x(7)	

Algorithm for Pairs

Multiply 2nd entry in Pair #1 by the weight for Pair #1 then ADD and SUBTRACT this product from the 1st entry in Pair #1.

Repeat for each pair

Level 2: Two 4-pt FFTs



$(W_8)^0 = 1\angle 0^o$ Use this weight for Pair #1 & #2

$(W_8)^1 = 1\angle{-45^o}$

$(W_8)^2 = 1\angle{-90^o}$ Use this weight for Pair #3 & #4

$(W_8)^3 = 1\angle{-135^o}$

Pair 1: $x(0) + x(4)$, $x(2) + x(6)$

Pair 2: $x(1) + x(5)$, $x(3) + x(7)$

Pair 3: $x(0) - x(4)$, $x(2) - x(6)$

Pair 4: $x(1) - x(5)$, $x(3) - x(7)$

Weighted Adds:

$[x(0)+x(4)]+1[x(2)+ x(6)]$

$[x(1)+x(5)]+1[x(3)+ x(7)]$

$[x(0)-x(4)]+(W_8)^2[x(2)- x(6)]$

$[x(1)-x(5)]+(W_8)^2[x(3)- x(7)]$

Weighted Subtracts:

$[x(0)+x(4)]-1[x(2)+ x(6)]$

$[x(1)+x(5)]-1[x(3)+ x(7)]$

$[x(0)-x(4)]-(W_8)^2[x(2)- x(6)]$

$[x(1)-x(5)]-(W_8)^2[x(3)- x(7)]$

Algorithm for Pairs

Multiply 2nd entry in Pair #1 by the weight for Pair #1 then ADD and SUBTRACT this product from the 1st entry in Pair #1.

Repeat for each pair

Level 3: One 8-pt FFTs

$(W_8)^0 = 1\angle 0^o$	Weight for Pair #1
$(W_8)^1 = 1\angle -45^o$	Weight for Pair #2
$(W_8)^2 = 1\angle -90^o$	Weight for Pair #3
$(W_8)^3 = 1\angle -135^o$	Weight for Pair #4

[x(0)+x(4)]+1[x(2)+ x(6)]	Pair 1
[x(1)+x(5)]+1[x(3)+ x(7)]	
[x(0)-x(4)]+$(W_8)^2$[x(2)- x(6)]	Pair 2
[x(1)-x(5)]+$(W_8)^2$[x(3)- x(7)]	
[x(0)+x(4)]–1[x(2)+ x(6)]	Pair 3
[x(1)+x(5)]–1[x(3)+ x(7)]	
[x(0)-x(4)]-$(W_8)^2$[x(2)- x(6)]	Pair 4
[x(1)-x(5)]-$(W_8)^2$[x(3)- x(7)]	

{[x(0)+x(4)]+1[x(2)+ x(6)]}+ 1{[x(1)+x(5)]+1[x(3)+ x(7)]}	Weighted Adds
{[x(0)-x(4)]+$(W_8)^2$[x(2)- x(6)]}+ $(W_8)^1${[x(1)-x(5)]+$(W_8)^2$[x(3)- x(7)]}	
{[x(0)+x(4)]–1[x(2)+ x(6)]}+ $(W_8)^2${[x(1)+x(5)]–1[x(3)+ x(7)]}	
{[x(0)-x(4)]-$(W_8)^2$[x(2)- x(6)]}+ $(W_8)^3${[x(1)-x(5)]-$(W_8)^2$[x(3)- x(7)]}	
{[x(0)+x(4)]+1[x(2)+ x(6)]}– 1{[x(1)+x(5)]+1[x(3)+ x(7)]}	Weighted Subtracts
{[x(0)-x(4)]+$(W_8)^2$[x(2)- x(6)]}– $(W_8)^1${[x(1)-x(5)]+$(W_8)^2$[x(3)- x(7)]}	
{[x(0)+x(4)]–1[x(2)+ x(6)]}– $(W_8)^2${[x(1)+x(5)]–1[x(3)+ x(7)]}	
{[x(0)-x(4)]-$(W_8)^2$[x(2)- x(6)]}– $(W_8)^3${[x(1)-x(5)]-$(W_8)^2$[x(3)- x(7)]}	

Algorithm for Pairs

Multiply 2nd entry in Pair #1 by the weight for Pair #1 then ADD and SUBTRACT this product from the 1st entry in Pair #1.

Repeat for each pair

Programming an 8-Point FFT

Memory Allocation:

- 1 8-element data buffer for input (re-ordered)
- 1 8-element data buffer for intermediate results
- 1 4-element Array of Weights (Need only half due to 180^o symmetry)

Processing:

Level 1: Each pair (N/2 of them) requires one multiply (4 multiplies)
Level 2: Each pair (N/2 of them) requires one multiply (4 multiplies)
Level 3: Each pair (N/2 of them) requires one multiply (4 multiplies)

Total Multiplies = (N/2)*Number of Levels = $(N/2)\log_2(N) = 12$

Comments

- This handout is a generic illustration of an FFT algorithm. Actual programming depends on a number of issues including the type of DSP processor used. For example, the 6711 DSK can do 8 multiplications in a single clock cycle so several pairs of data could be processed simultaneously.

- Remember that an FFT is simply a computationally efficient method of computing a Discrete Fourier Transform. An FFT will give the same results as a straight DFT with significantly fewer multiplications!

N	**Number of Multiplications**	
	DFT = N^2	**FFT = $(N/2)\log_2(N)$**
8	64	12
32	1024	80
256	65,536	1024
512	262,144	2304

Conclusion

Digital signal processing is a challenging course to teach to engineering technology students due to the level of mathematics involved. Students have access to FFT functions and blocks through MATLAB and SIMULINK as well as FFT algorithms already developed for the TI DSK evaluation boards used in lab. They could easily use these tools with little or no understanding of the FFT algorithm. However, I feel it is important for them to understand the tools they are using to develop a real appreciation for what DSP is all about: processing signals quickly and efficiently to achieve some practical objective(s).

The traditional textbook approach to teaching FFTs is to present students with a complicated set of equations showing how data is separated into pairs, processed using a 2-pt FFT, then recombined with another 2-pt FFT. The pairs of 2-pt FFTs are then processed to create a set of 4-pt FFTs and the process is repeated until one single N-pt FFT has been computed. The equations are typically followed with a Butterfly diagram used to illustrate the flow of data. While the idea is easy to convey to engineering technology students, the equations tend to be overwhelming for most of my students. The handout described in this paper was presented to my DSP students last year following a lecture using the traditional approach. My students felt that they understood how an FFT algorithm works much better after seeing the handout. In fact, I had a couple of typos in the equations on the handout which they were able to catch and correct convincing me that they were indeed getting it. Using the handout made it much easier to teach FFT concepts and applications to my engineering technology students.

Bibliography

[1] Digital Signal Processing: A Practical Guide for Engineers and Scientist, by Steven W. Smith, Newnes, 2002.

[2] Signal Processing First, by James McClellan, Ronald Schafer, and Mark Yoder, Prentice Hall, 2003.

[3] Digital Signal Processing: A Practical Approach, by Emmanuel Ifeachor and Barrie Jervis, Prentice Hall, 2002.

[4] Discrete-Time Signal Processing, by Alan V. Oppenheim, Ronald Schafer, John R. Buck, Prentice Hall, 1999.

[5] Analog and Digital Signal Processing, by Ashok Ambardar, Brooks/Cole Publishing Co., 1999.

[6] Digital Signal Processing, by Jack Cartinhour, Prentice Hall, 2000.