NATIONAL RADIO ASTRONOMY OBSERVATORY
GREEN BANK, WEST VIRGINIA

SPECTRAL PROCESSOR MEMO NO. 23

MEMORANDUM

January 3, 1985

To: Spectral Processor Project Group

From: R. Fisher, R. Weimer

C: R. Stokes

Subj: Hardware Notes

This memo puts two sets of handwritten notes into the spectral processor memo series since the diagrams and equations that they contain will probably be referred to in the future. These notes were originally used as discussion points in past design meetings and do not contain any narrative explanations.

The first note gives the decimation in time FFT butterfly multiplier and adder configurations, and the second derives the functional diagram of the correction for the use of a complex FFT on real input data.

JRF/RBW/cjd

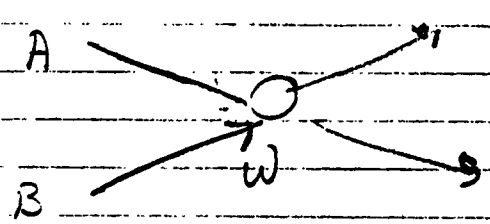
Attachments

1. R. Weimer, "Butterfly Equations and Functional Diagrams"
2. R. Fisher, "Real Correction Equations and Functional Diagrams"

R Weimer

Butterfly equations and functional diagrams

USING D.I.T.



$$W = e^{-j\theta}$$

$$A = A_R + j A_I$$

$$B = B_R + j B_I$$

$$W = \cos\theta - j \sin\theta$$

$$C = A_R + j A_I + (B_R + j B_I)(\cos\theta - j \sin\theta)$$

$$= A_R + B_R \cos\theta + B_I \sin\theta + j(A_I + B_I \cos\theta - B_R \sin\theta)$$

$$C_R = A_R + B_R \cos\theta + B_I \sin\theta$$

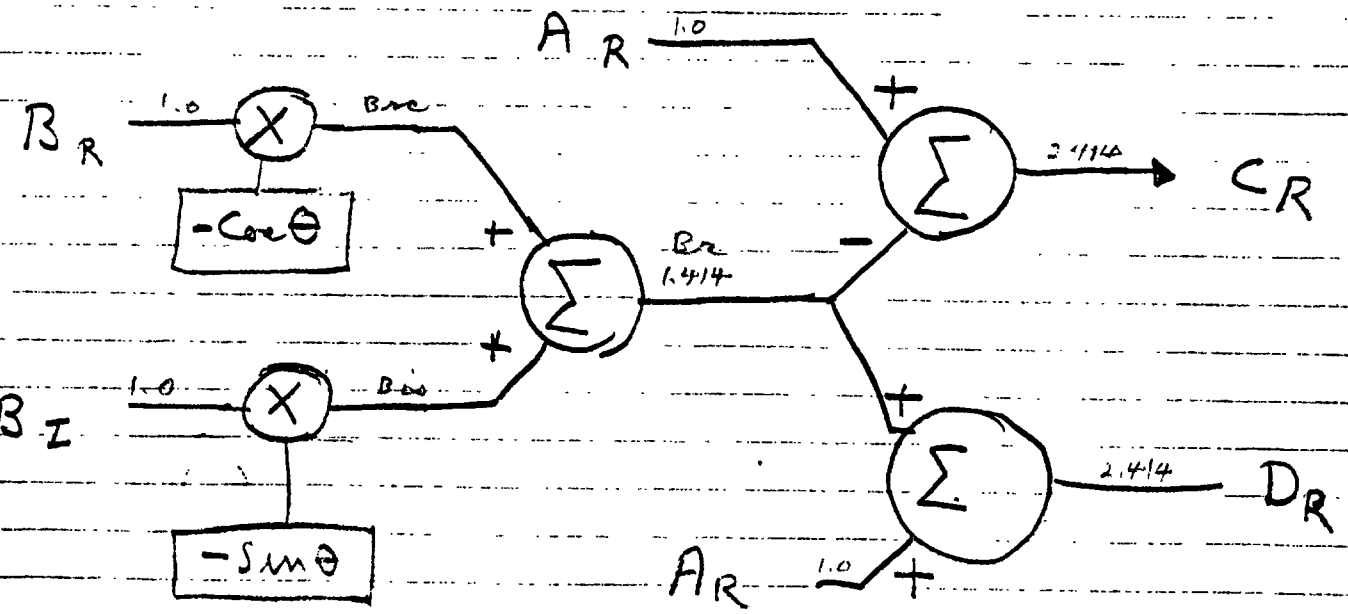
$$C_I = A_I + B_I \cos\theta - B_R \sin\theta$$

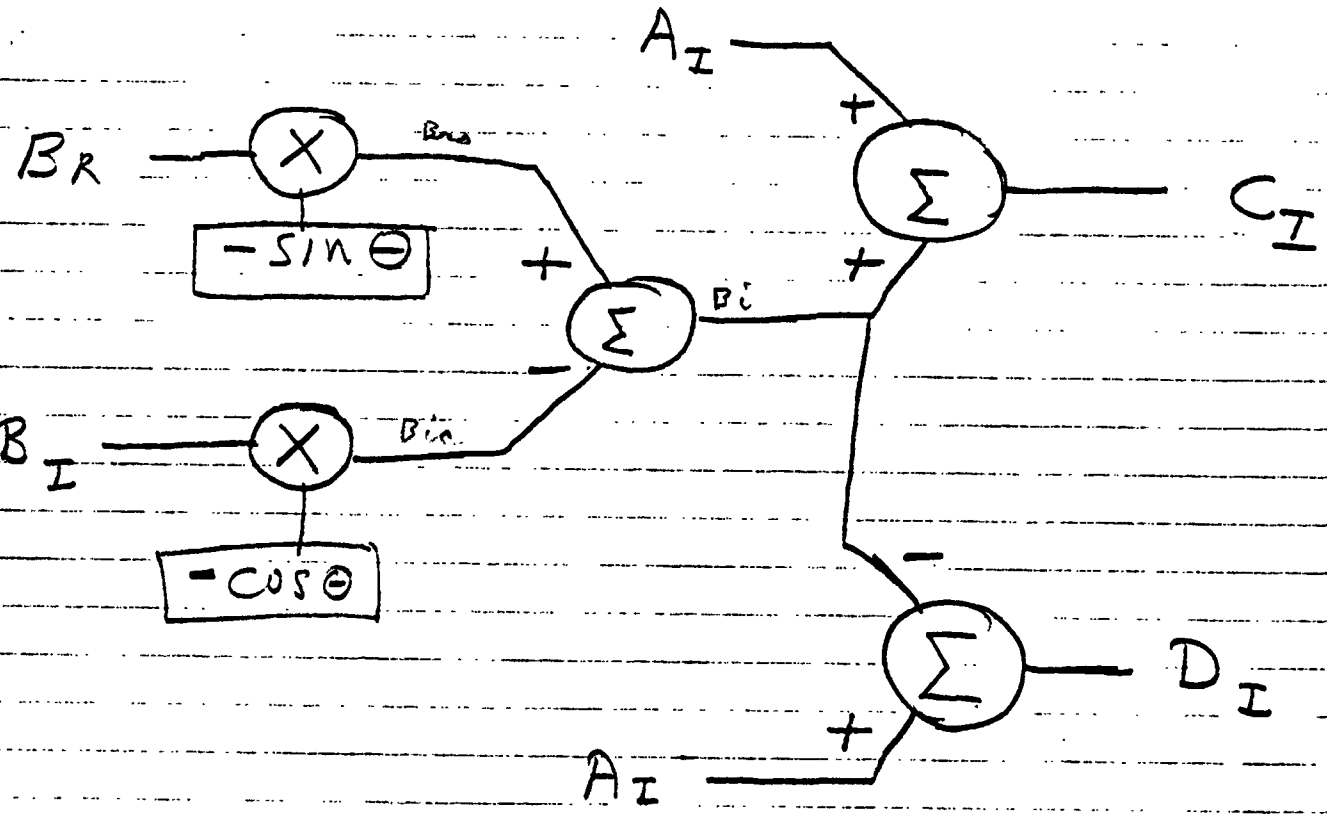
$$D = A_R + jA_I - (B_R + jB_I)(\cos\theta - j\sin\theta)$$

$$= A_R - B_R\cos\theta - B_I\sin\theta + j(A_I - B_I\cos\theta + B_R\sin\theta)$$

$$D_R = A_R - B_R\cos\theta - B_I\sin\theta$$

$$D_I = A_I - B_I\cos\theta + B_R\sin\theta$$





DATA IN AVAIL
⊖ TO ROM

CLOCK INTO
MULT

CLOCK OUT
OF MULT

LATCH OUT OF
ENABLE A_R & A_I to ACC 2

LATCH OUT
OF SECOND

R. Fisher

Real correction equations + functional diagrams

M = # of complex channels = # of real points / 2

$C(k)$ = k^{th} complex output of FFT

$G(k)$ = k^{th} complex frequency channel

$$G(k) \propto \left(C(k) + C^*(M-k)\right) - i\, W^{\frac{k}{2M}} \left(C(k) - C^*(M-k)\right)$$

$$G(M-k) \propto \left(C(M-k) + C^*(k)\right) - i\, W^{\frac{(M-k)}{2M}} \left(C(M-k) - C^*(k)\right)$$

$$W^{\frac{k}{2M}} = \cos\frac{\pi k}{M} - j \sin\frac{\pi k}{M} \qquad k = 0 \ldots M-1$$

$$W^{\frac{M-k}{2M}} = \cos\frac{\pi(M-k)}{M} - j \sin\frac{\pi(M-k)}{M}$$

$$= \cos\left(\pi - \frac{\pi k}{M}\right) - j \sin\left(\pi - \frac{\pi k}{M}\right)$$

$$= -\cos\left(\frac{\pi k}{M}\right) - j \sin\left(\frac{\pi k}{M}\right)$$

$$= -W^{\frac{k}{2M}*}$$

$$G(M-k) \propto \left(C(M-k) + C^*(k)\right) + i\, W^{\frac{k}{2M}*} \left(C(M-k) - C^*(k)\right)$$

let $A = C(k) = A_R + iA_I$

$B = C(M-k) = B_R + iB_I$

$W^{\frac{k}{2M}} = W_R \mp i\, W_I = \cos\left(\frac{\pi k}{M}\right) - i \sin\left(\frac{\pi k}{M}\right)$

$C = G(k)$

$D = G(M-k)$

} "butterfly outputs

$$C = A + B^* - i W^{\frac{\hbar}{2m}} (A - B^*)$$

$$D = B + A^* + i W^{\frac{\hbar}{2m}*} (B - A^*)$$

$$C = A_R + iA_I + B_R - iB_I - i(W_R - iW_I)(A_R + iA_I - B_R + iB_I)$$

$$D = B_R + iB_I + A_R - iA_I + i(W_R + iW_I)(B_R + iB_I - A_R + iA_I)$$

$$C = A_R + B_R + i(A_I - B_I) - i(W_R A_R - W_R B_R + W_I A_I + W_I B_I$$
$$- iW_I A_R + iW_I B_R + iW_R A_I + iW_R B_I$$

$$D = B_R + A_R + i(B_I - A_I) + i(W_R B_R - W_R A_R - W_I B_I - W_I A_I$$
$$+ iW_R B_R - iW_I A_R + iW_R B_I + iW_R A_I)$$

$$C = (A_R + B_R) + i(A_I - B_I) - i(W_R A_R - W_R B_R + W_I A_I + W_I B_I)$$
$$+ (-W_I A_R + W_I B_R + W_R A_I + W_R B_I)$$

$$D = B_R + A_R + i(B_I - A_I) + i(W_R B_R - W_R A_R - W_I B_I - W_I A_I)$$
$$- W_I B_R + W_I A_R - W_R B_I - W_R A_I$$

$$-i(A_I - B_I)$$

$$D = A_R + B_R - (-W_I A_R + W_I B_R + W_R A_I + W_R B_I)$$
$$- i(W_R A_R - W_R B_R + W_I A_I + W_I B_I)$$

Revised
5 Dec 84

$$C_R = (A_R + B_R) + (W_R A_I + W_R B_I - W_I A_R + W_I B_R)$$

$$D_R = (A_R + B_R) - (W_R A_I + W_R B_I - W_I A_R + W_I B_R)$$

$$C_I = -(W_R A_R - W_R B_R + W_I A_I + W_I B_I) + (A_I - B_I)$$

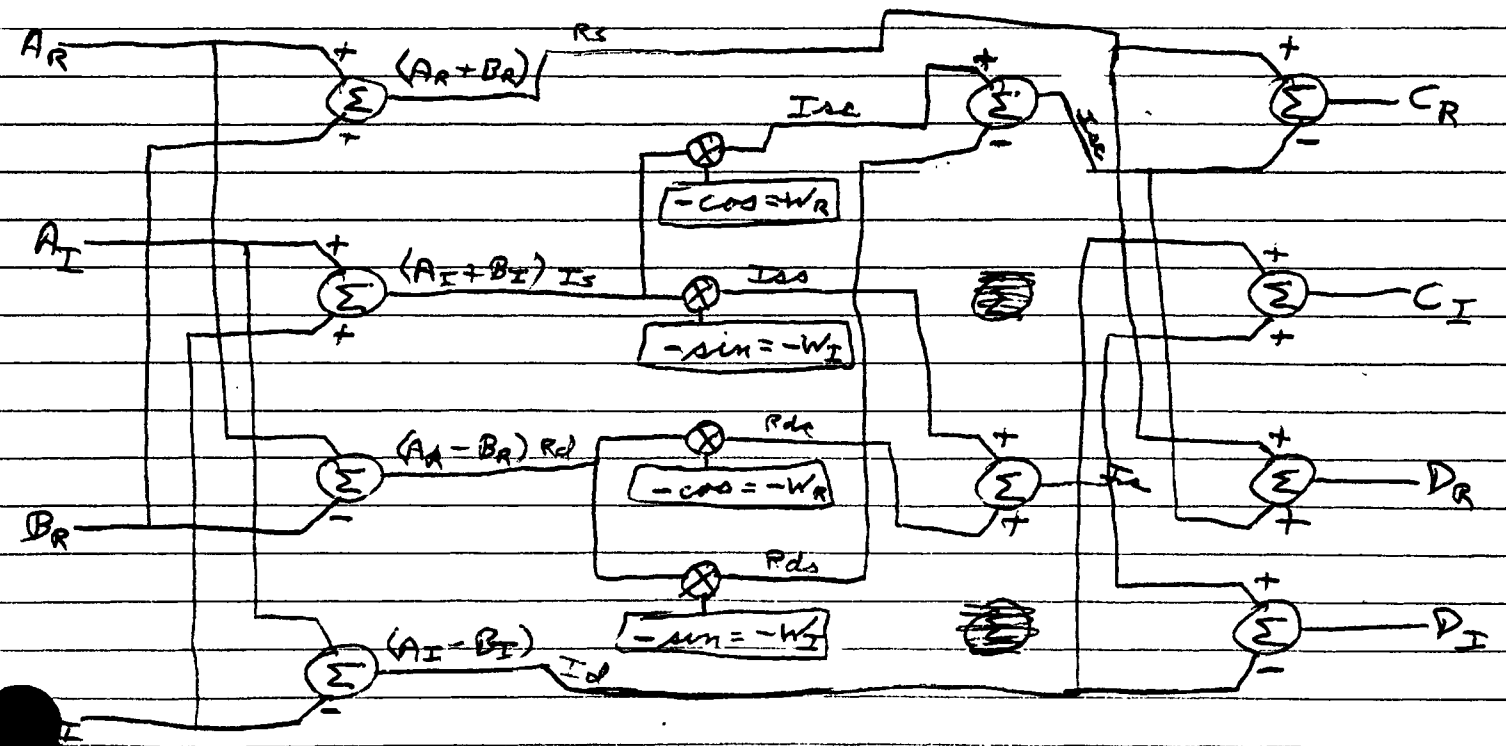
$$D_I = -(W_R A_R - W_R B_R + W_I A_I + W_I B_I) - (A_I - B_I)$$

$$C_R = (A_R + B_R) + W_R(A_I + B_I) - W_I(A_R - B_R)$$

$$D_R = (A_R + B_R) - W_R(A_I + B_I) + W_I(A_R - B_R)$$

$$C_I = (A_I - B_I) - W_R(A_R - B_R) - W_I(A_I + B_I)$$

$$D_I = -(A_I - B_I) - W_R(A_R - B_R) - W_I(A_I + B_I)$$



same for every partial transformation. This is a major advantage if the FFT is to be implemented in hardware; similarly, Singleton[2] organizes his FFT of tape-stored data around this rule. The disadvantages of this rule are that the data must be permuted initially, and that at least $N[1 - (1/f_a)]$ elements of working storage are needed.

6.2.1.4. The Transform of Data with Many Zeros. Occasionally, a small number N_1 of input data are to be transformed to a large number N_2 of output values. The standard technique is to append $N_2 - N_1$ zeros to the input and use the normal FFT. However, many of the operations are superfluous since they operate on zero data. When both N_1 and N_2 are powers of two, Markel[3] has given a simple trick which reduces the running time by a factor of approximately $\log_2 N_2/[(\log_2 N_1) + 1]$. Program FOUR1 may be modified to achieve this speedup by inserting the new statement

```
ISTEP=MIN0(ISTEP,IMAX/N1)
```

immediately following the definition of ISTEP. The total number of data in FOUR1, called N, is, of course, N_2; N_1, called N1 above, should be included in the calling sequence.

Similarly, if only the first N_3 ($\ll N_2$) output values are desired, a time factor of $\log_2 N_2/[(\log_2 N_3) + 1]$ may be saved by changing the statement

```
DO 80 I1=1,ITWOL,IP0
```

to

```
I1MAX=MIN0(ITWOL,IP0*N3)
DO 80 I1=1,I1MAX,IP0
```

Similar changes apply to the other programs.

6.2.1.5. The Transform of Real Data. If the input $g(j)$ is real, then the transform $G(k)$ is complex; but only $(N/2) + 1$ values of $G(k)$ are independent for, from Eq. (6.2.3), the complex conjugate

$$G^*(k) = G(-k), \tag{6.2.31}$$

where $-k$ is taken modulo N. Therefore, it is sufficient to retain in storage only $G(0)$ through $G(N/2)$. These complex values take up slightly more storage than the N real values input from $g(j)$.

If N, the number of input data, is even, a very simple trick may be used to transform N real data in half the space and half the time of N complex data. First, from the real array $g(j)$, construct a complex array $c(j)$ of half the length by

$$c(j) = g(2j) + ig(2j + 1), \qquad j = 0, \ldots, (N/2) - 1. \tag{6.2.32}$$

[2] R. C. Singleton, *IEEE Trans. Audio Electroacoust.* AU15, 91 (1967).
[3] J. Markel, *IEEE Trans. Audio Electroacoust.* AU19, 305 (1971).

Now, transform $c(j)$ into $C(k)$:

$$C(k) = 1/(N/2)^{1/2} \sum_{j=0}^{(N/2)-1} c(j)1^{jk/(N/2)}$$

$$= (2/N)^{1/2} \sum_{j=0}^{(N/2)-1} [g(2j) + ig(2j+1)]1^{2jk/N}, \qquad k = 0, \ldots, (N/2) - 1.$$

Notice that $C(N/2) = C(0)$. Changing k to $(N/2) - k$ and conjugating, we have

$$C^*((N/2) - k) = (2/N)^{1/2} \sum_{j=0}^{(N/2)-1} [g(2j) - ig(2j+1)]1^{2jk/N}.$$

Then, from Eq. (6.2.3),

$$G(k) = 1/N^{1/2} \sum_{j=0}^{(N/2)-1} [g(2j)1^{2jk/N} + g(2j+1)1^{(2j+1)k/N}]$$

$$= (1/2\sqrt{2})[(C(k) + C^*((N/2) - k)) - i1^{k/N}(C(k) - C^*((N/2) - k))]. \tag{6.2.33}$$

Program FXRL1 in Appendix D implements Eqs. (6.2.32) and (6.2.33). An alternate, more complicated way of transforming real data is given by Bergland.[4]

6.2.1.6. The Transform of Multidimensional Data. As shown above, a two-dimensional FFT [Eq. (6.2.6)] may be performed as a set of in-place, one-dimensional transforms on the rows and columns [Eqs. (6.2.7) and (6.2.8)]. Programs CFFT2, RFFT2, and HFFT2 in Appendix E implement Eq. (6.2.6) when transforming complex arrays, real arrays, and transforms of real arrays, respectively. An $N_1 \times N_2$ real array is transformed into an $((N_1/2) + 1) \times N_2$ complex array, since

$$G^*(k_1, k_2) = G(-k_1, -k_2), \tag{6.2.34}$$

taking the subscripts modulo N_1 and N_2.

Faster versions of the FFT can be written by restricting the number of data to a power of two and factoring by fours. This gives a 25% speedup over factoring by twos. Bergland[4] points out that factoring by eights is 10% faster yet, but the program size quadruples. Further speedup can be obtained by computing complex exponentials by Singleton's recursion.[5] The author has written a program named FOUR2 that incorporates factoring by four and use of the above recursion. This program uses in-place mapping and therefore requires no working storage at all. Copies of this program may be obtained from the author upon request.

[4] G. D. Bergland, *IEEE Trans. Audio Electroacoust.* AU17, 138 (1969).
[5] R. C. Singleton, *IEEE Trans. Audio Electroacoust.* AU17, 93 (1969).