AD-787 502

IMAGE TRANSMISSION VIA SPREAD SPECTRUM TECHNIQUES

Robert W. Means, et al

Naval Undersea Center

Prepared for:

Advanced Research Projects Agency

1974

ARPA QR4

IMAGE TRANSMISSION VIA SPREAD SPECTRUM TECHNIQUES

ARPA Quarterly Technical Report

January 2, 1974 – July 1, 1974

Advanced Research Projects Agency
Order Number 2303
Code Number 3G10

Naval Undersea Center
San Diego, California 92132

IMAGE TRANSMISSION
VIA
SPREAD SPECTRUM TECHNIQUES

Investigated by
DR. ROBERT W. MEANS (714:225-6872), Code 608
JEFFREY M. SPEISER (714:225-6607), Code 60_1
and
HARPER J. WHITEHOUSE (714:225-6315), Code 6003
Naval Undersea Center
San Diego, California

Sponsored by
Advanced Research Projects Agency
Order Number 2303
Code Number 3G10
Contract
effective: 15 February 1973
expiration: 30 June 1974
amount: $519,400

ARPA Quarterly Technical Report

January 1, 1974 – July 1, 1974

Form Approved Budget Bureau No. 22-R0293

ia

ABSTRACT

This report addresses the design of a spread spectrum image transmission system to provide increased antijam protection to a television link from a small remotely piloted vehicle. Previous quarterly reports have described component developments, theoretical advances in image processing, and simulations of proposed coding schemes. This report summarizes the proposed total system.

CONTENTS

REPORT SUMMARY

This report describes the proposed system for bandwidth reduction of the video for a remotely piloted vehicle. Two systems are being constructed: one compatible with a standard vidicon, the other compatible with a Charge Injection Device 100 x 100 camera. The transversal filters for the 100 point Cosine Transform were delivered during this phase and integrated into a transform subsystem. A limited prototype ground station was designed and constructed during this phase. The proposed transform was shown to be relatively tolerant to bit errors, which is shown in Appendix A. The proposed coding schemes were investigated and are reported in Appendix B. The modularity of the CHIRP-Z Transform is shown in Appendix C. This modularity and the structures described in Appendix D allow general purpose high-speed signal processors which could be used in the channel decoder.

INTRODUCTION

The Image Transmission via Spread Spectrum Techniques program has developed to a point where a complete proposed system can be described. Previous quarterly reports have described component developments, theoretical advances in image processing, and simulations of proposed coding schemes. A summary of the complete system description was presented during an ARPA workshop at MITRE on May 14-15, 1974. This summary is presented below. Also included are Appendixes A thrugh F which present study efforts conducted during this last reporting period by individual authors.

Appendix A is a compilation of photographs of transformed images done at the University of Southern California. Appendix B is a comparison of methods of channel encoding. Appendix C is a description of the modular CHIRP-Z Transform. Appendix D is a description of a high capacity linear processor. Appendix E is a description of the utilization of a signal processing imager chip. Appendix F is an updated description of a signal processing interpreter for use in a general purpose computer.

SYSTEM DESCRIPTION

The following material is presented in summary format and represents the status of the complete proposed program.

Problem Statement

Design and demonstrate a real time image bandwidth reduction and spread spectrum encoding system for use in a lightweight Remotely Piloted Vehicle (RPV) television system to provide protection against intentional and unintentional jamming.

Goals for RPV Image Redundancy Reduction System

a) ~ 1.5 lb

b) 15 - 30 watts

c) Field demonstration in spring 1975.

Approach

Reduce data rate in order to use spread spectrum encoding

- Number of image points
- Frame rate
- Transform encoding

Performance of Intraframe Encoding

PCM/DPCM

8 – 3 bits/pixel

Transform with

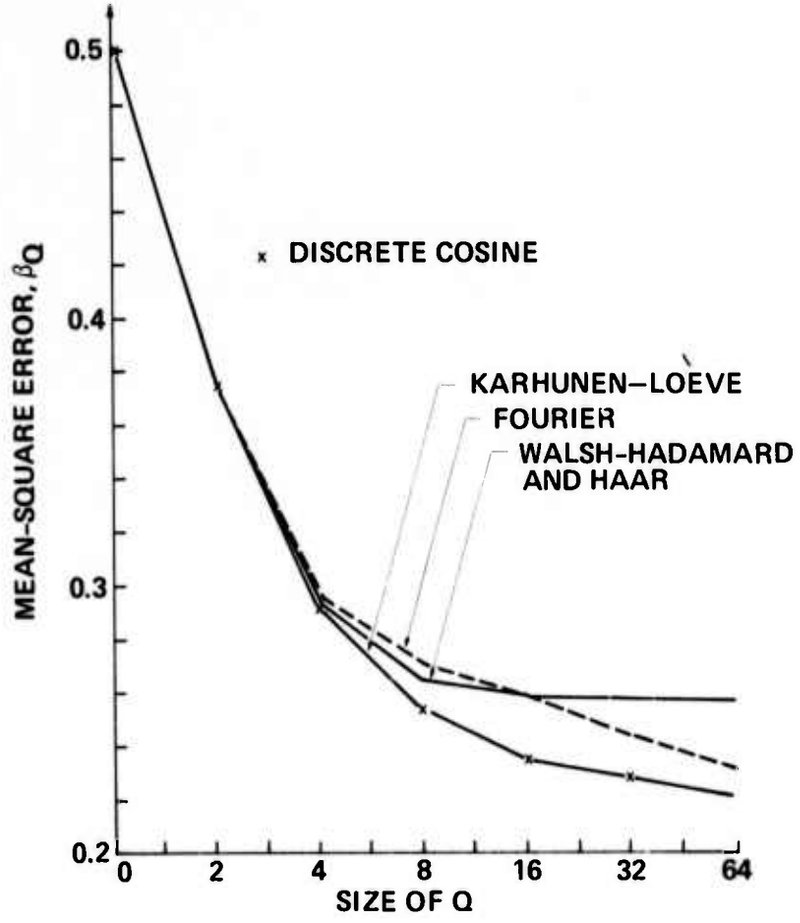
Selective: 3 – 1 bits/pixel

Quantization

Transforms Examined:

- Haar
- Hadamard
- Slant
- Fourier
- Differential Pulse Code Modulation (DPCM)
- Discrete Cosine Transform (DCT)
- Karhunen - Loève (K - L)

Mean Square Error Performance of Various Transforms for Exponentially Correlated Data, $\rho = 0.9$:



Discrete Fourier Transform, CZT Algorithm:

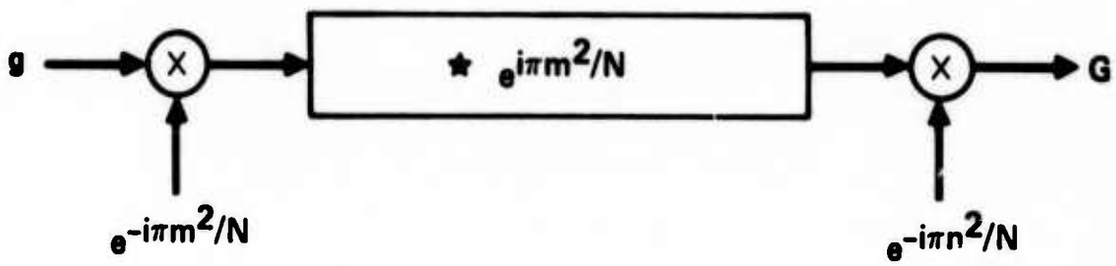
1) $G_m = \sum_{n=O}^{N-1} e^{-j2\pi mn/N} g_n$

2) $-2mn = -m^2 + (m-n)^2 - n^2$

3) $\therefore G_m = e^{-j\pi m^2/N} \sum_{n=O}^{N-1} e^{j\pi(m-n)^2/N} (e^{-j\pi n^2/N} g_n)$

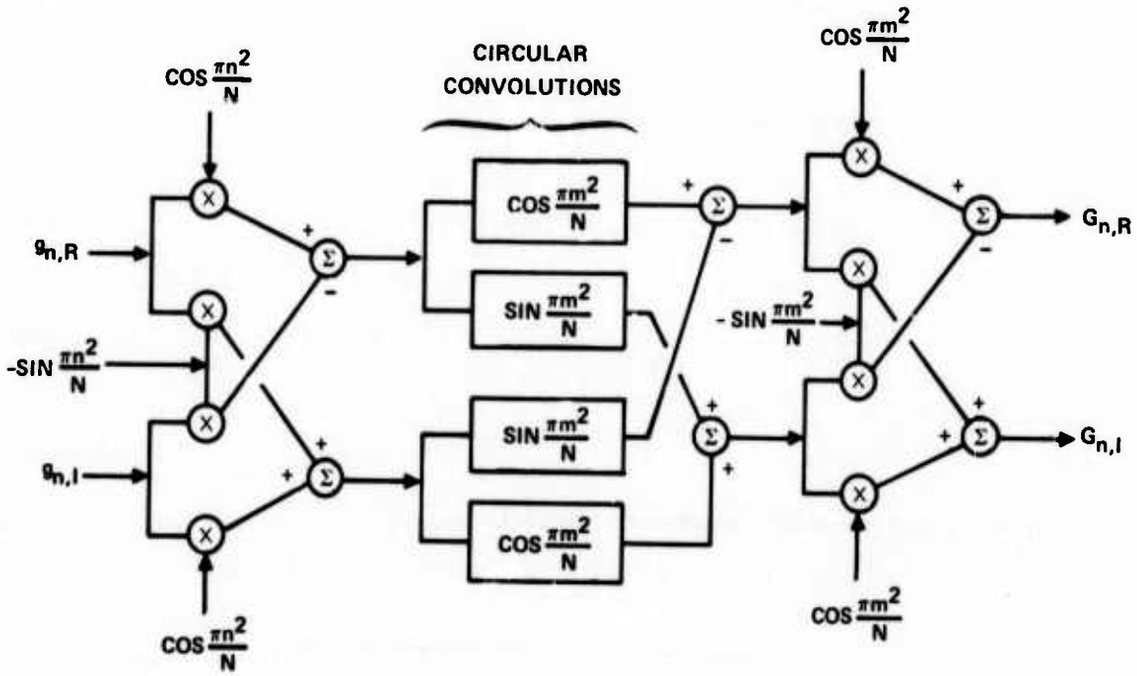
where $m = 0, 1, 2, \ldots, N-1$.

CHIRP-Z Transform Implementation of the DFT

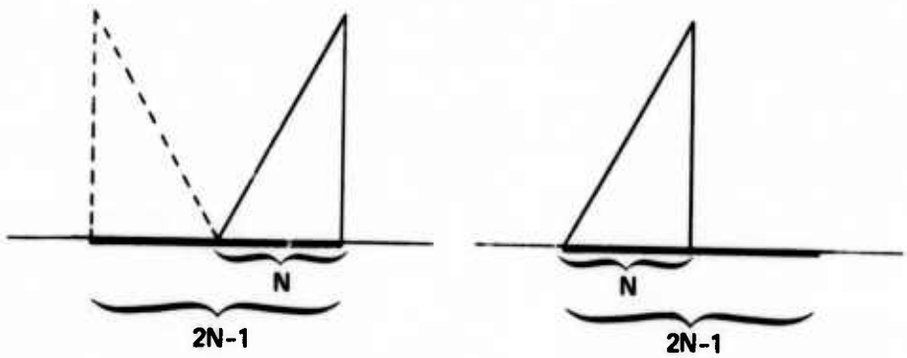


★ Denotes either convolution or circular convolution

DFT Via CZT Algorithm with Parallel Implementation of Complex Arithmetic

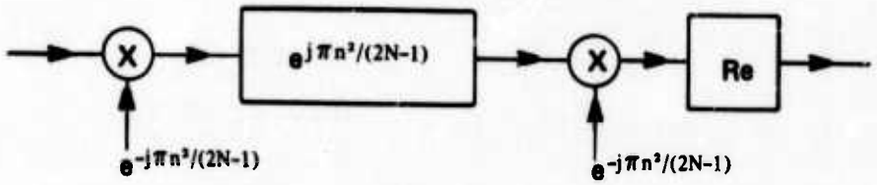


Relation of DCT to DFT



DCT = DFT of symmetric extension = real part of DFT of null extension

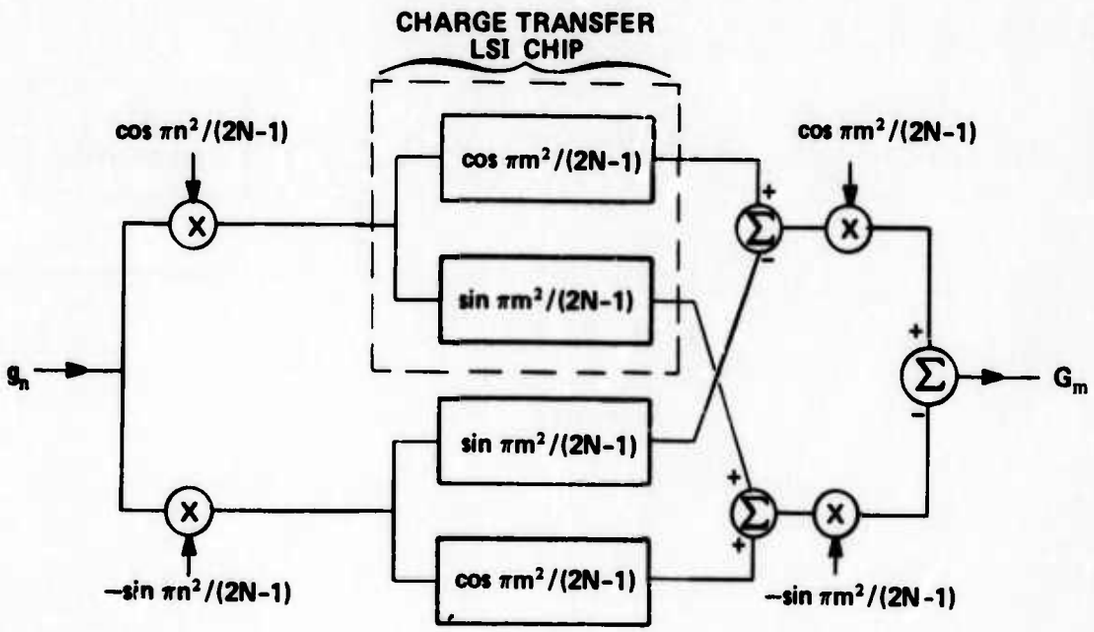
Serial Access Implementation of the DCT



$-(N-1) \leq n \leq (N-1)$

$0 \leq n \leq (N-1)$ $\qquad$ $0 \leq n \leq (N-1)$

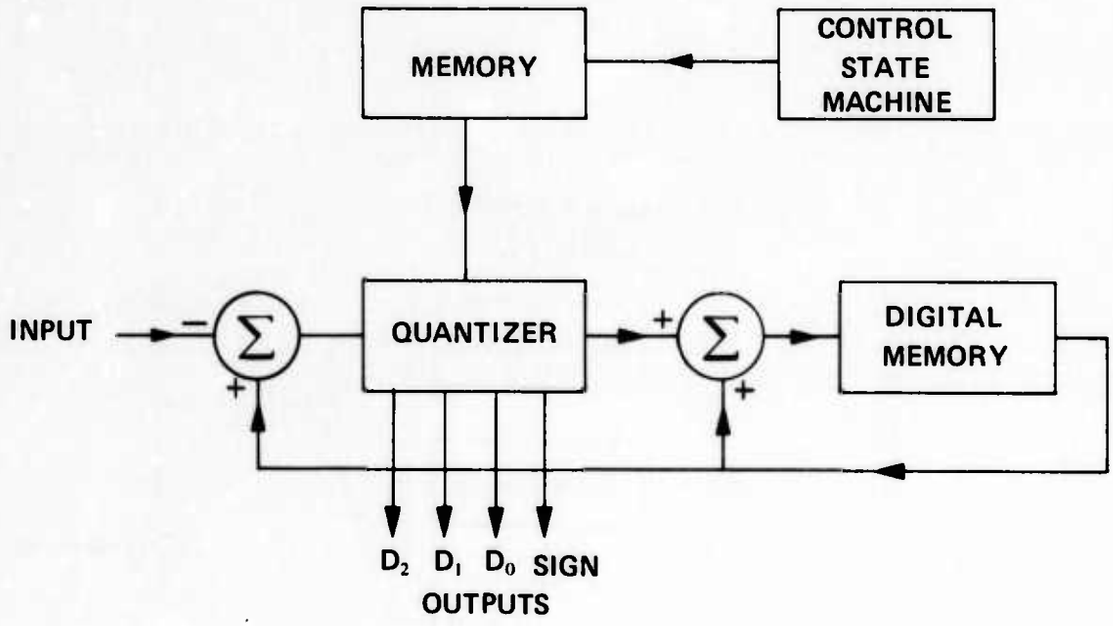
Charge Transfer Implementation of DCT



Transforms Examined

Two - Dimensional	Hybrid
Haar	Haar/DCT
Hadamard	Hadamard/DPCM
Fourier	Fourier/DPCM
DPCM	DCT/DPCM
DCT	

DPCM Encoder



10^5 Samples/Second Transform System

- 100 X 100 CID sensor
- 100 point BBD implementation of DCT/DPCM

5 X 10^6 Samples/Second Transform System

- 256 X 240 sensor
- 32 point CCD implementation of DCT/DPCM

Coding Design

Source

a) minimum of 4 frames/sec

b) 256 X 256 pixels

c) mean square error $\approx$ 30 dB down

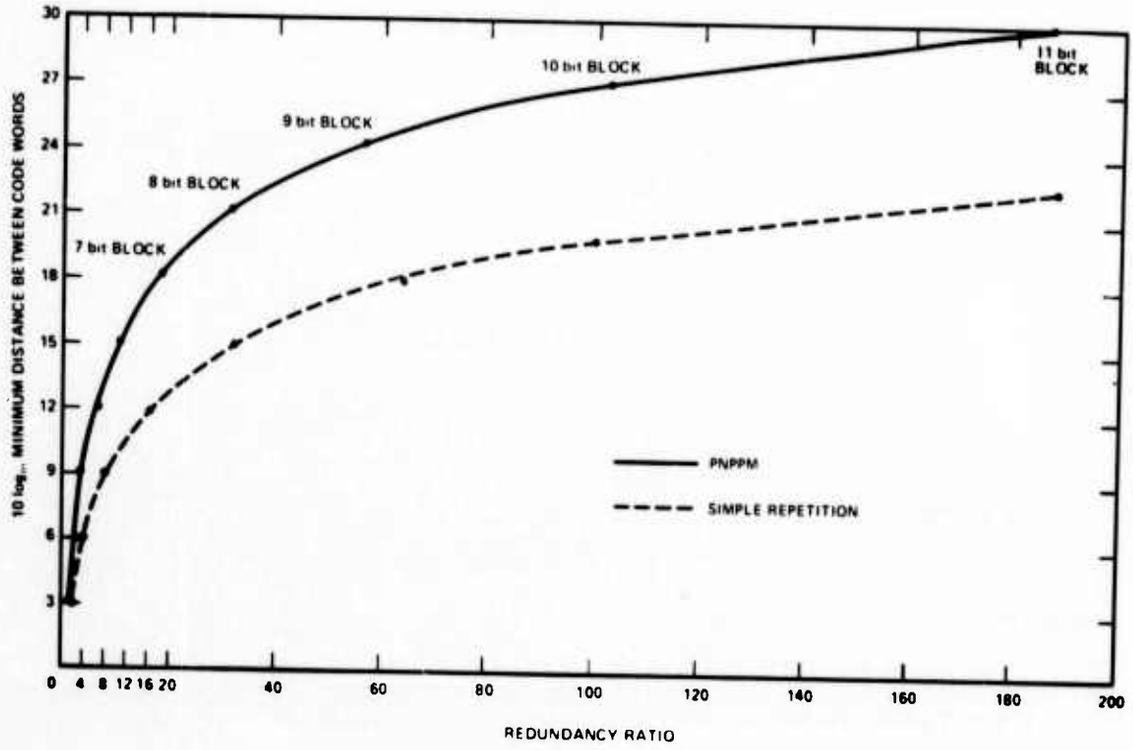
Channel

a) 20 Mbits/sec

b) at each data rate maximize distance between code words

c) coding for modem independence

Performance of PNPPM and Simple Repetition as a Function of Redundancy Ratio



System Performance for a Linear Matched Filter PN-PPM Receiver

FRAMES/ SEC	BITS/PIXEL					
30	1	.56	.31			
15	1.75	1	.56	.31		
7-1/2	3	1.75	1	.56	.31	
3-3/4		3	1.75	1	.56	
1-7/8			3	1.75	1	
15/16				3	1.75	
	18	21	24	27	30	dB

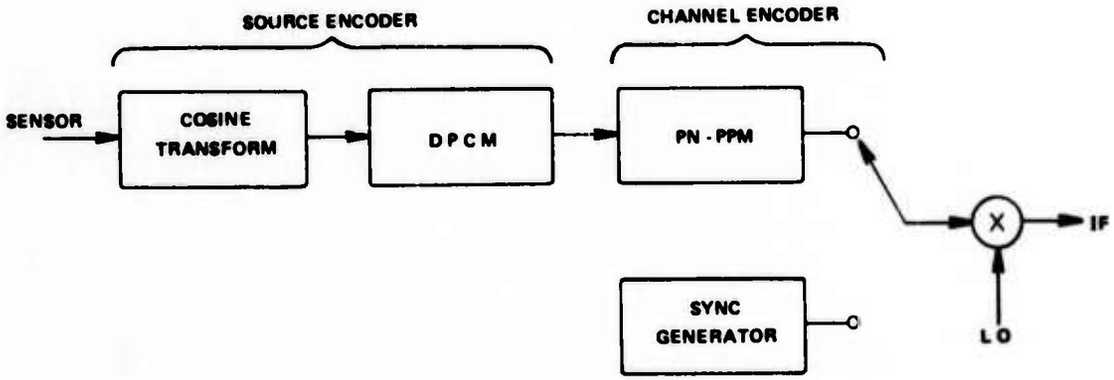
MATCHED FILTER PROCESSING GAIN

Performance of Proposed System

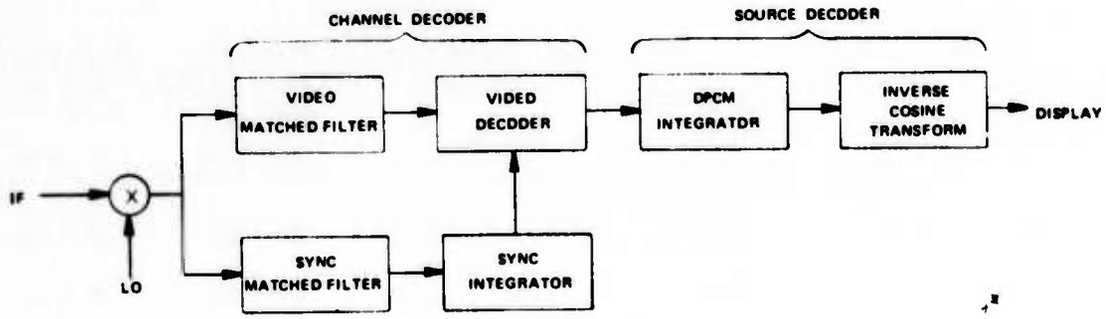
CODE WORD LENGTH	*BITS PER PIXEL*	*COMPRESSION RATIO*	*GAUSSIAN NOISE MARGIN*		*PULSED INTERFERENCE MARGIN* (CODE WORD SEPARATION)	*GAUSSIAN NOISE MARGIN* OVER SPEC*
			$P = 10^{-2}$	$P = 10^{-3}$		
1023	0.31	102.3	22.4 dB	23.6 dB	30.1 dB	27.7 dB
511	0.56	56.8	19.7 dB	20.5 dB	27.1 dB	25.0 dB
255	1.0	32.0	17.2 dB	18.1 dB	24.1 dB	22.5 dB

*$P = 10^{-5}$
S/N = 10 dB

Transmitter



Receiver



APPENDIX A

UNIVERSITY OF SOUTHERN CALIFORNIA
TRANSFORM SIMULATION

TRANSFORMED IMAGES

This appendix presents a compilation of photographs of several image coding systems based upon transform coding performed at the University of Southern California.

Cosine/DPCM, P = 0

2 Dim. Cosine, P = 0

Cosine/DPCM, P = 10^{-2}

2 Dim. Cosine, P = 10^{-2}

Original

Hadamard/DPCM, P = 0

Figure 1. Coded Pictures, 2 Bits/Pixel.

Cosine/DPCM, P = 0

2 Dim. Cosine, P = 0

Cosine/DPCM, P = 10^{-2}

2 Dim. Cosine, P = 10^{-2}

Hadamard/DPCM, P = 0

Figure 2. Coded Pictures, 1 Bit/Pixel.

Cosine/DPCM, P = 0

2 Dim. Cosine, P = 0

Cosine/DPCM, $P = 10^{-2}$

2 Dim. Cosine, $P = 10^{-2}$

Hadamard/DPCM, P = 0

Figure 3. Coded Pictures, 0.5 Bit/Pixel.

original

Cosine/DPCM, P = 0

Cosine/DPCM, P = 10^{-3}

Cosine/DPCM, P = 10^{-2}

Figure 4. Coded Pictures, 2 Bits/Pixel.

original

Cosine/DPCM, P = 0

Cosine/DPCM, $P = 10^{-3}$

Cosine/DPCM, $P = 10^{-2}$

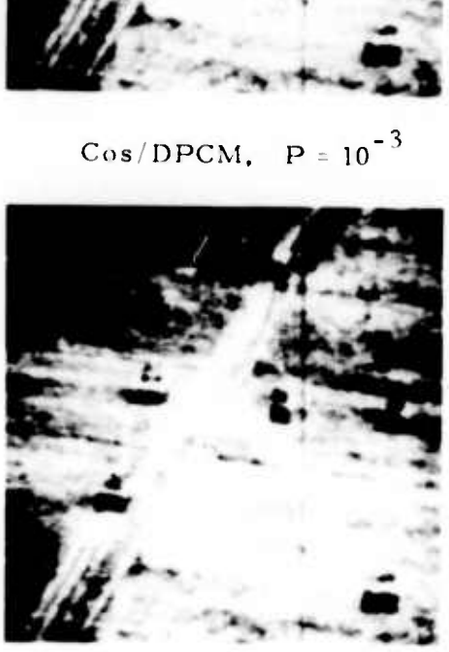
Figure 5. Coded Pictures, 2 Bits/Pixel.

Cos/DPCM, P = 0

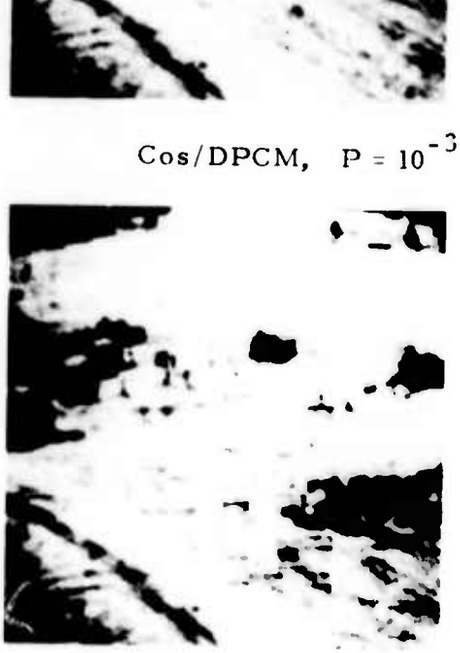
Cos/DPCM, P = 0

Cos/DPCM, P = 10^{-3}

Cos/DPCM, P = 10^{-3}



Cos/DPCM, P = 10^{-2}



Cos/DPCM, P = 10^{-2}

Figure 6. Coded Pictures, 1 Bit/Pixel.

Cos/DPCM, P = 0

Cos/DPCM, P = 0

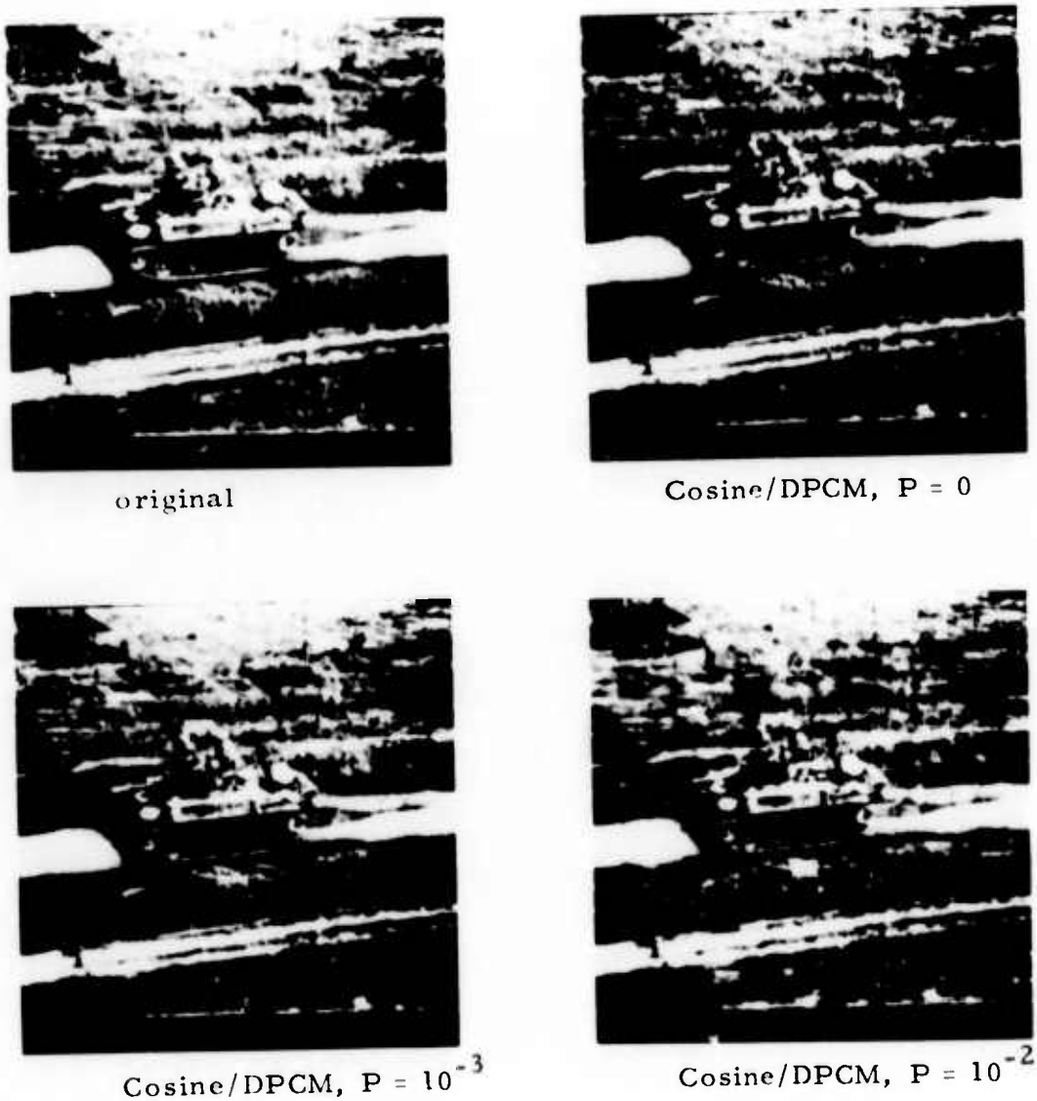
Cos/DPCM, P = 10^{-3}

Cos/DPCM, P = 10^{-3}

Cos/DPCM, P = 10^{-2}

Cos/DPCM, P = 10^{-2}

Figure 7. Coded Pictures, 0.5 Bit/Pixel.



original

Cosine/DPCM, P = 0

Cosine/DPCM, P = 10^{-3}

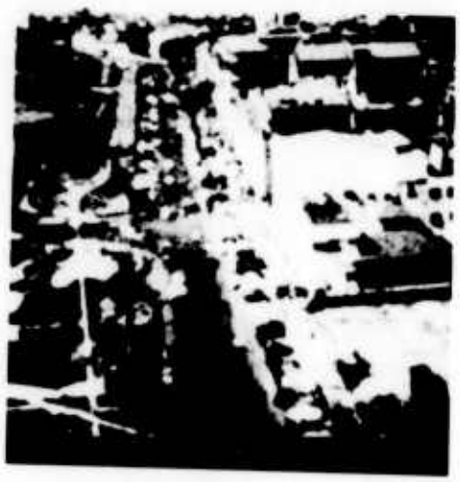
Cosine/DPCM, P = 10^{-2}

Figure 8. Coded Pictures, 2 Bits/Pixel.

original

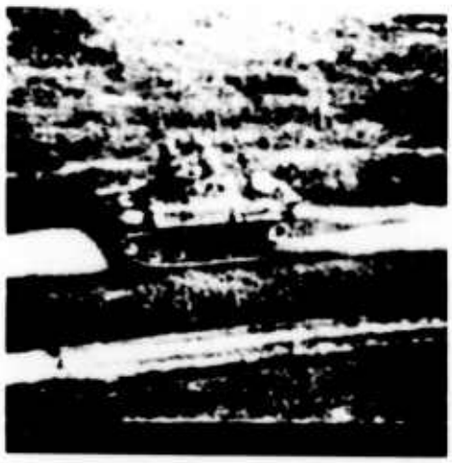
Cosine/DPCM, P = 0

Cosine/DPCM, P = 10^{-3}



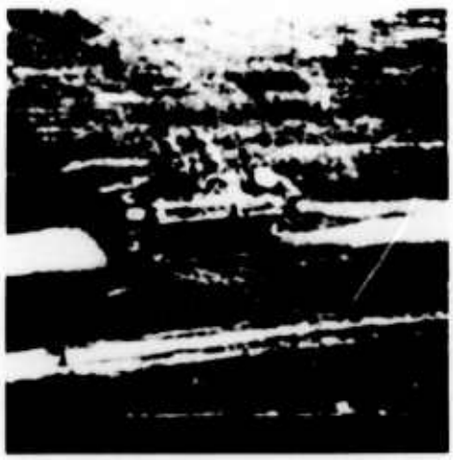
Cosine/DPCM, P = 10^{-2}

Figure 9. Coded Pictures, 2 Bits/Pixel.



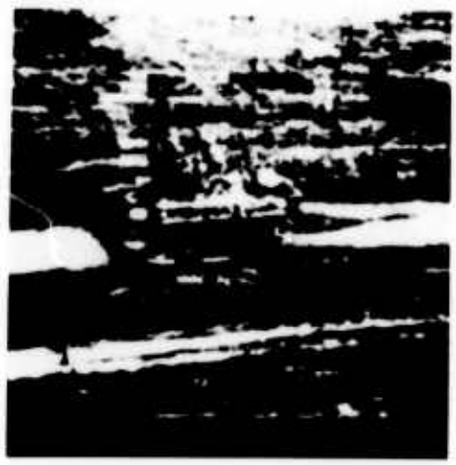
Cos/DPCM, P = 0

Cos/DPCM, P = 0



Cos/DPCM, P = 10^{-3}

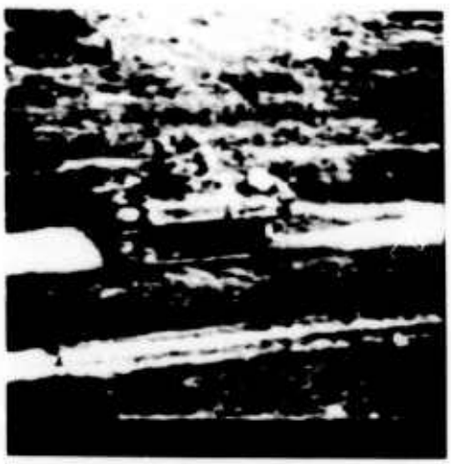
Cos/DPCM, P = 10^{-3}



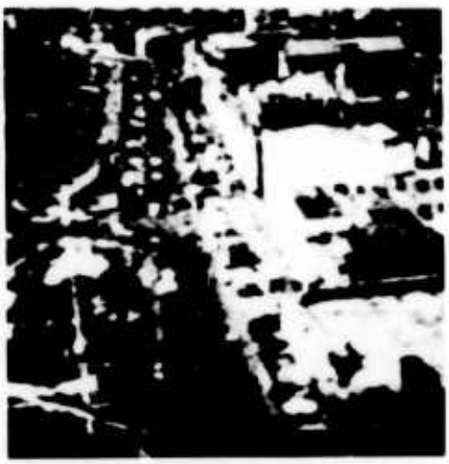
Cos/DPCM, P = 10^{-2}

Cos/DPCM, P = 10^{-2}

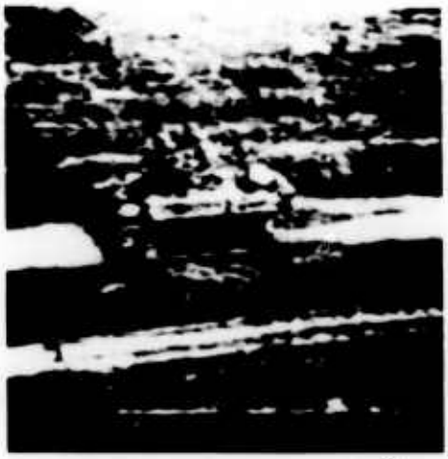
Figure 10. Coded Pictures, 1 Bit/Pixel.



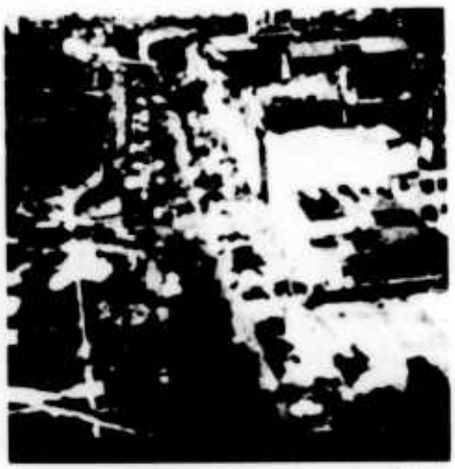
Cos/DPCM, P = 0



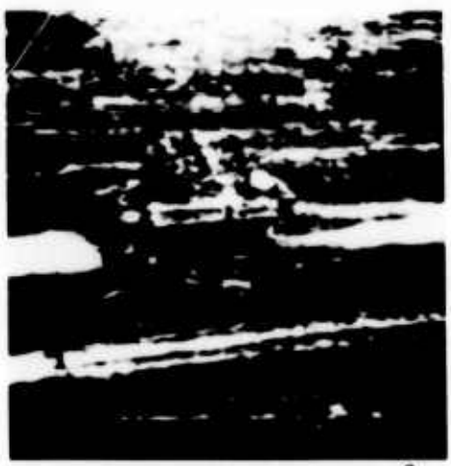
Cos/DPCM, P = 0



Cos/DPCM, P = 10^{-3}



Cos/DPCM, P = 10^{-3}



Cos/DPCM, P = 10^{-2}

Cos/DPCM, P = 10^{-2}

Figure 11. Coded Pictures, 0.5 Bit/Pixel.

APPENDIX B

VIDEO SPREAD SPECTRUM ENCODING

Dr. E. H. Wrench, Jr.

Naval Undersea Center

VIDEO SPREAD SPECTRUM ENCODING

by

Edwin H. Wrench, Jr.

INTRODUCTION

In the design of a communication system there is usually a trade off between system performance and complexity (expense). Such trade offs between complexity and performance in the presence of noise exist for the video down link for the ARPA remotely piloted vehicle. The allocated channel bandwidth is barely adequate to transmit the video directly; however, if direct transmission is used, noise immunity can only be obtained by increased transmitted power. For a system of limited size, weight, and power dissipation, this approach is undesirable.

An alternative approach is to use bandwidth compression techniques to remove redundancy from the video prior to transmission. The compressed video can then be transmitted directly or recoded to fill the allocated channel bandwidth. The latter technique further increases noise immunity by trading bandwidth for increased signal-to-noise ratio. This more complicated system offers the same performance as the direct transmission scheme but with reduced transmitted power.

In this paper the theoretical performance of three proposed video down link communication systems are compared with the performance of a reference system which uses direct transmission of the raw video data. The first system (system 1) transmits the binary compressed video directly using biorthogonal code words. The second system (system 2)

uses M-ary signaling techniques to increase the bandwidth of the compressed video to that of the original raw video data. The code words are pseudorandom noise (PN) sequences generated by maximal length shift register code generating techniques. They have the property that the auto correlation function has constant side lobes of $-1/M$, where M is the length of the code word. In the receiver, the received signal is correlated with M code references. The correlator with the largest output is chosen as the transmitted code word (matched filter detection). The third system (system 3) uses code words that are identical to those of system 2, but uses binary transmission with biorthogonal code words. The M bits of the code words are then detected on a bit-to-bit basis and the resulting detected bit stream fed into the correlators. As with system 2, the code word corresponding to the largest correlator output is chosen as the transmitted code word. The difference between systems 2 and 3 is that system 3 detection is performed prior to correlation rather than using correlation to perform the detection as in system 2.

The performance of all three systems will be compared with that of the reference system (system 4), which transmits and detects a 20 megabit per second video on a bit-to-bit basis. The code words used are biorthogonal. The noise is assumed to be Gaussian distributed white noise in all cases and the bit error rate taken as 10^{-2}.

Several different compression ratios for the video were examined. The compression ratios for various length PN sequences and for a frame rate reduction factor of eight are given in Table 1.

Table 1. Compression Ratio for Various Code Lengths for a Frame Rate Reduction Factor of Eight

PN Length, "M"	*Compression Ratio*
255	32
511	56.8
1023	102.4

Analysis of the Reference System (System 4)

The signal-to-noise ratio γ at the output of the matched filter required to achieve a given bit error rate in a binary system with biorthogonal code words is

$$\gamma = [\text{erfc}^{-1} (2 P_b)]^2.$$

For a bit error rate (P_b) of 10^{-2},

$$\gamma \approx 2.64 = 4.3 \text{ dB}.$$

Analysis of System 1

System 1 is identical to the reference system (system 4) except the transmission rate has been reduced by a compression ratio factor (C), where

$$C = \frac{\text{No. of bits/sec original data}}{\text{No. of bits/sec compressed data}}.$$

The output signal-to-noise ratio of the matched filter is

$$\gamma = \int_{-\infty}^{\infty} \frac{S(t)\, dt}{N_o},$$

where S(t) is the instantaneous received signal power, and N_o is the noise spectral density.

Since the duration of the signal pulse has increased for system 1 by a factor of C,

$$\int_{-\infty}^{\infty} S_1(t)dt = C\int_{-\infty}^{\infty} S_4(t)dt.$$

Therefore the signal-to-noise ratio for system 1 is increased by a factor of C over that of system 4. The improvement in signal-to-noise ratio for 3 values of C are given in Table 2.

Table 2. Signal-to-Noise Ratio Improvement (dB) for System 1

Compression Ratio (C)	*Improvement (dB)*
32	15.1
56.8	17.5
102.4	20.1

Analysis of System 2

The performance of System 2, employing M-ary signaling with matched filter detection, is described by the probability of a symbol error,

$$P_s = 1 - \int_{-\infty}^{\infty} (2\pi)^{-1/2} e^{-x^2/2} \left[\int_{-\infty}^{x+(2\gamma)^{1/2}} (2\pi)^{-1/2} e^{-y^2/2} dy \right]^{m-1} dx.$$

For large M, probability of a bit error is

$$P_b \approx 1/2 P_s.$$

The values of P_s for various values of M and γ are tabulated by Viterbi.[1]

The improvement in the required signal-to-noise ratio is given in Table 3 for various compression ratios.

---

[1] Viterbi, Andrew J. *Phase-Coherent Communication over Continuous Gaussian Channel in Digital Communications with Space Applications* by Solomon W. Golomb. Prentice Hall, 1964.

Therefore the signal-to-noise ratio for system 1 is increased by a factor of C over that of system 4. The improvement in signal-to-noise ratio for 3 values of C are given in Table 2.

Table 2. Signal-to-Noise Ratio Improvement (dB) for System 1

Compression Ratio (C)	*Improvement (dB)*
32	15.1
56.8	17.5
102.4	20.1

Analysis of System 2

The performance of System 2, employing M-ary signaling with matched filter detection, is described by the probability of a symbol error,

$$P_s = 1 - \int_{-\infty}^{\infty} (2\pi)^{-1/2} e^{-x^2/2} \left[\int_{-\infty}^{x+(2\gamma)^{1/2}} (2\pi)^{-1/2} e^{-y^2/2} dy \right]^{m-1} dx.$$

For large M, probability of a bit error is

$$P_b \approx 1/2 P_s.$$

The values of P_s for various values of M and γ are tabulated by Viterbi.[1]

The improvement in the required signal-to-noise ratio is given in Table 3 for various compression ratios.

---

[1] Viterbi, Andrew J. *Phase-Coherent Communication over Continuous Gaussian Channel in Digital Communications with Space Applications* by Solomon W. Golomb. Prentice Hall, 1964.

Table 3. Signal-to-Noise Ratio Improvement (dB) for System 2

Compression Ratio (C)	*Improvement (dB)*
32	17.2
56.8	19.7
102.4	22.4

Analysis of System 3

System 3 uses matched filtering following detection on a bit-to-bit basis. The probability of a symbol error is derived as follows: There are M correlators, one for each code word. The output from the correlator which corresponds to the transmitted code has a value of

$$M_E = M - 2E$$

when exactly E bit errors have occurred. The correlators with only side lobe sums have values of

$$S_E = -1 - 2A + 2D,$$

where A is the number of bit errors in bits that should agree with the reference, and D is the number of bit errors in bits that should disagree with the reference. Since A + D = E, the equation can be rewritten

$$S_E = -1 + 4D - 2E.$$

For an individual correlator, an error will occur if the side lobe sum exceeds the main lobe sum, i.e., $S_E > M_E$. There is also a possible error when $S_E = M_E$. The probability of making an error in any correlator, given E bit errors, is

$$P_{c/E} = \text{Prob}\,(S_E > M_E) + 1/2\ \text{Prob}\,(S_E = M_E).$$

But, $\text{Prob}\,(S_E > M_E) = \text{Prob}\,(-1-4A-2E > M-2E) = \text{Prob}\,(D > \frac{M+1}{4}) \Big|$ No. errors = E

so that $$P_{c/E} = \text{Prob}\,(D > \frac{M+1}{4}) \Big|_E + 1/2\ \text{Prob}\,(D = \frac{M+1}{4}) \Big|_E,$$

or, $$P_{c/E} = 1\text{-HYGEO}\,(\frac{M+1}{4}, M, \frac{M-1}{2}, E) + 1/2\ [\text{HYGEO}\,(\frac{M+1}{4}, M, \frac{M-1}{2}, E) - \text{HYGEO}\,(\frac{M-3}{4}, M, \frac{M+1}{2}, E)],$$

where HYGEO is the hypergeometric distribution

$$H(K,N,N_1,N_R) = \sum_{i=0}^{K} \binom{N_R}{i} \binom{N-N_R}{N_1-i} \Big/ \binom{N}{N_1}.$$

If we assume the probability of an error in a correlator is independent of the error in the other correlators, the total probability of a correlator error is

$$P_c = (1\text{-Prob. of no correlator errors}) = 1 - (1 - P_{c/E})^{(M-1)}.$$

The probability of a symbol error can then be written as

$$P_s = \sum_{E=0}^{M} \text{Prob}\,(E) \cdot P_c = \sum_{E=[(M+1)/4]}^{M} \text{Prob}\,(E) \cdot P_c,$$

where Prob (E) is the probability of E errors

$$\text{Prob}\,(E) = (P)^E\,(1-P)^{M-E} \binom{M}{E},$$

and where P is the probability of a single bit error (determined by S/N) in the detector. For biorthogonal coherent detection

$$P = 1/2 \operatorname{erfc} \sqrt{\gamma}\,.$$

Solution of the equation for P_s yields the results shown in Figure 1. For M = 255, and $P_b = 0.01 = 1/2\, P_s$, the required signal-to-noise ratio is -10.8 dB. This same bit error probability is obtained without bandwidth compression and coding with S/N ≈ 4.2 dB. The improvement is then 15.0 dB for the compressed coded video over straight video transmission. These results and those for M = 511 and M = 1023 are given in Table 4.

Table 4. Signal-to-Noise Ratio Improvement (dB) for System 3 ($P_b = 10^{-2}$)

Compression Ratio (C)	*Improvement (dB)*
32	15.5
56.8	18.0
102.4	20.8

Discussion

The analysis results are summarized for the three systems in Table 5 for a probability of a bit error equal to 10^{-2}. Similar results are given for a probability of a bit error equal to 10^{-3} in Table 6. Table 7 summarizes the improvement for System 2 with pulse interference rather than Gaussian noise.

Each of the three systems examined has advantages and disadvantages. System 1 gives a 15-20 dB improvement over the reference system (system 4) and requires no sophisticated coding or decoding. However, it is a relatively narrow band system, depending on the compression ratio, that lends itself to easy detection by a jammer.

System 2 gives a 17.2 - 23.7 dB improvement, and wide bandwidth, at the expense of greatly increased system complexity, since it requires matched filters at IF frequencies. System 3 is a compromise between system 1 and system 4. The transmission and detection are done one bit at a time to eliminate the need for large analog matched filters. The performance is within 2 dB of the optimum, provided by system 2.

Table 5. Summary of System Improvement (dB) with the Probability of a Bit Error Equal to 10^{-2} and Gaussian Noise

System	*Compression Ratio (C)* 32	56.8	102.4
1	15.1	17.5	20.1
2	17.2	19.7	22.4
3	15.5	18.0	20.8

Table 6. Summary of System Improvement (dB) with the Probability of a Bit Error Equal to 10^{-3} and Gaussian Noise

System	*Compression Ratio (C)* 32	56.8	102.4
1	15.1	17.5	20.1
2	18.4	21.0	23.7
3	16.4	19.3	22.1

Table 7. System 2 Improvement (dB) with Pulse Noise (10 Log [Code Word Separation])

Compression Ratio (C)	32	56.8	102.4
Improvement (dB)	21	24	27

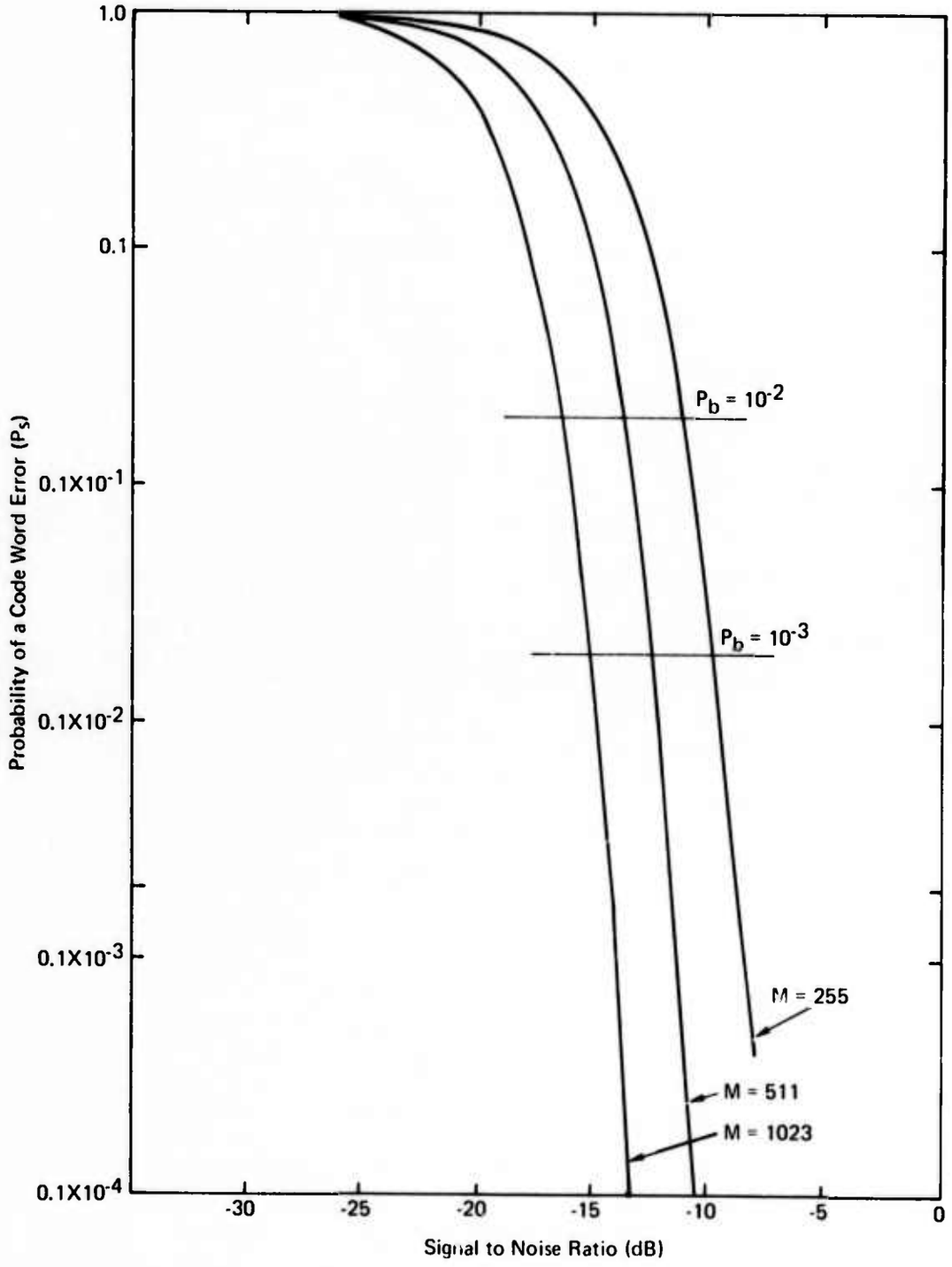


Figure B-1. System 3 Performance

APPENDIX C

THE MODULAR CHIRP–Z TRANSFORM

J. M. Speiser and H. J. Whitehouse

Naval Undersea Center

ABSTRACT

Two methods for combining P Chirp-Z Transform (CZT) modules of length N to perform a Discrete Fourier Transform (DFT) of length NP are described. The firs method uses an auxiliary parallel-input, parallel-output DFT device of size P and allows the transform of size NP to be performed in the same time required fo a single CZT module to perform a size N transform. The second method uses an auxiliary parallel-input, serial-output DFT device of size P. If the second method is implemented entirely in a single technology, such as with CCD's, it performs the size NP transform in P times the amount of time required for a single CZT module to perform a size N transform; if P is a composite number, say $P = P_1P_2$, the second method also permits the same hardware to perform P_1 simultaneous transforms of length NP_2.

INTRODUCTION

Many signal processing problems require flexible real-time implementations of linear signal processing operations such as Fourier transforms, convolution correlation, and beamforming. All of these operations may be performed at high throughput rates using the discrete Fourier transform implemented via the CZT algorithm with a transversal filter or cross convolver used to perform the required convolution or correlation with a discrete chirp [1-3] as shown in Fig. 1 and described by equations (1) - (3). Special purpose methods have also been previously described for combining a number of CZT modules to perform a longer DFT [4]. The previous methods, however, required a different acoustic surface wave filter for each different number of CZT modules to be combined. This limits the flexibility of the transform size attainable with a given set of components, and also prevents the longer transform system from being externally clocked, since the propagation time through the surface wave device cannot be varied by more than a small fraction of one percent.

A one-dimensional discrete Fourier transform may be written as a partial transform of a doubly subscripted representation of the data, followed by a pointwise multiplication, followed by a second partial transform [4,5] as shown in equations (4) - (9).

$$H_k = \sum_{j=0}^{M-1} h_j \; e^{\frac{-i2\pi jk}{M}} \qquad (1)$$

$$H_k = P_k^* \sum_{j=0}^{M-1} [h_j P_j^*] P_{k-j} \quad (2)$$

$$P_s = e^{\frac{i\pi s^2}{M}} = P_{-s} \quad (3)$$

MODULAR CZT

$$G_k = \sum_{n=0}^{PN-1} g_n e^{\frac{-i2\pi kn}{PN}} \quad \text{for } k = 0, \ldots PN-1 \quad (4)$$

$$G_k = \sum_{s=0}^{P-1} \sum_{n=0}^{N-1} g_{Pn+s} e^{\frac{-i2\pi k(Pn+s)}{PN}} \quad (5)$$

$$G_k = \sum_{s=0}^{P-1} e^{\frac{-i2\pi ks}{PN}} \sum_{n=0}^{N-1} g_{Pn+s} e^{\frac{-i2\pi kn}{N}} \quad (6)$$

Let $Q_s(k) = e^{\frac{-i2\pi ks}{PN}}$ (7)

$$Q_s(k+nN) = e^{\frac{-i2\pi(k+nN)s}{PN}}$$

$$= e^{\frac{-i2\pi ks}{PN}} e^{\frac{-i2\pi nNs}{PN}} \quad (8)$$

$$Q_s(k+nN) = e^{\frac{-i2\pi sn}{P}} Q_s(k) = c_{sn} Q_s(k) \quad (9)$$

where $c_{sn} = e^{\frac{-i2\pi sn}{P}}$

Figures 2 and 2A show modular CZT implementations which follow from equations (4) - (9). The individual CZT subsystems shown in Figures 2 and 2A

would be similar to the CZT implementations previously described [1-3]. A suitable serial to parallel multiplexer is shown in Fig. 3.

The parallel DFT required for the second partial transform in Fig. 2 may be implemented as combination of summers and attenuators. This is shown in Figures 4-6 for P = 2, 3, and 4. In general, the attenuation factors are complex, requiring separate weightings for the real and imaginary parts as shown in Fig. 7. A complete double length CZT is shown in Fig. 8. Unfortunately, a parallel DFT implementation of this type becomes unwieldy if the dimension P is very large.

Other types of discrete Fourier transform implementations may be derived from the identities of equations (10) - (12).

$$H_k = \sum_{n=0}^{P-1} e^{\frac{-i2\pi kn}{P}} h_k \tag{10}$$

$$kn = \frac{1}{4}\left((k+n)^2 - (k-n)^2\right) \tag{11}$$

$$H_k = \sum_{n=0}^{P-1} e^{\frac{-i\pi(k+n)^2}{2P}} e^{\frac{i\pi(k-n)^2}{2P}} h_k \tag{12}$$

These equations have been used by Means [6] to design a transform device in which signals are shifted through two delay lines at different speeds. Alternatively, if the factors in equation (12) are interpreted as two waves propagating in opposite directions relative to the function to be transformed, it may be seen that the structure of Fig. 9 also performs a discrete Fourier transform with speed comparable to that of a CZT. By changing the reference functions applied to the delay lines and partitioning the input leads and output summer as shown

in Fig. 10, the hardware of Fig. 9 may also perform several shorter discrete Fourier transforms. If the parallel input-serial output CZT of Fig. 10 is used in the modular CZT of Fig. 2A, then P_1 discrete Fourier transforms of length P_2N may be performed simultaneously, where $P = P_1P_2$ is any factorization of P_1.

REFERENCES

[1] Whitehouse, H. J., J. M. Speiser, and R. W. Means, High Speed Serial Access Linear Transform Implementations, presented at the All Applications Digital Computer (AADC) Symposium, Orlando, Florida, 23-25 January 1973, reprinted as NUC TN 1026.

[2] Means, R. W., D. D. Buss, and H. J. Whitehouse, Real Time Discrete Fourier Transforms Using Charge Transfer Devices, Proceedings of the CCD Applications Conference held at the Naval Electronics Laboratory Center, San Diego, Calif., 18-20 Sept. 1973, pp. 95-101.

[3] Alsup, J. M., R. W. Means, and H. J. Whitehouse, Real-Time Discrete Fourier Transforms Using Surface Acoustic Wave Devices, Proceedings of the IEE International Specialist Seminar on Component Performance and System Applications of Surface Acoustic Wave Devices, held at Aviemore, Scotland, 24-28 Sept. 1973.

[4] Speiser, J. M., H. J. Whitehouse, and J. M. Alsup, High Capacity Linear Processing Using Multidimensional Chirp-Z Transforms, NUC TN 1212, 13 Nov. 1973.

[5] Gold, Ben and Theodore Bially, Parallelism in Fast Fourier Transform Hardware, IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No. 1, pp. 5-16, Feb. 1973.

[6] Means, R. W., Private Communication.

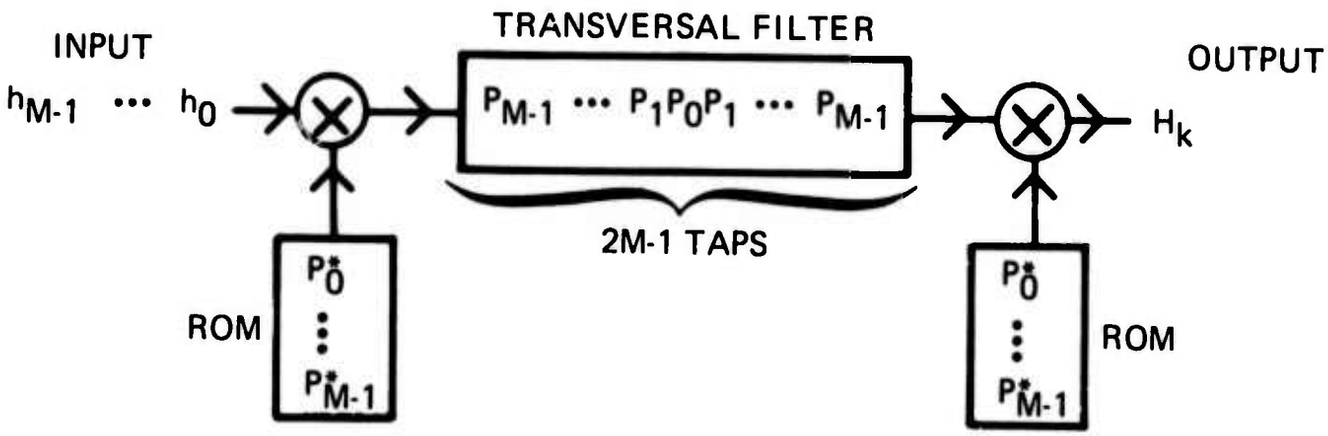


Figure 1. Direct CZT Implementation

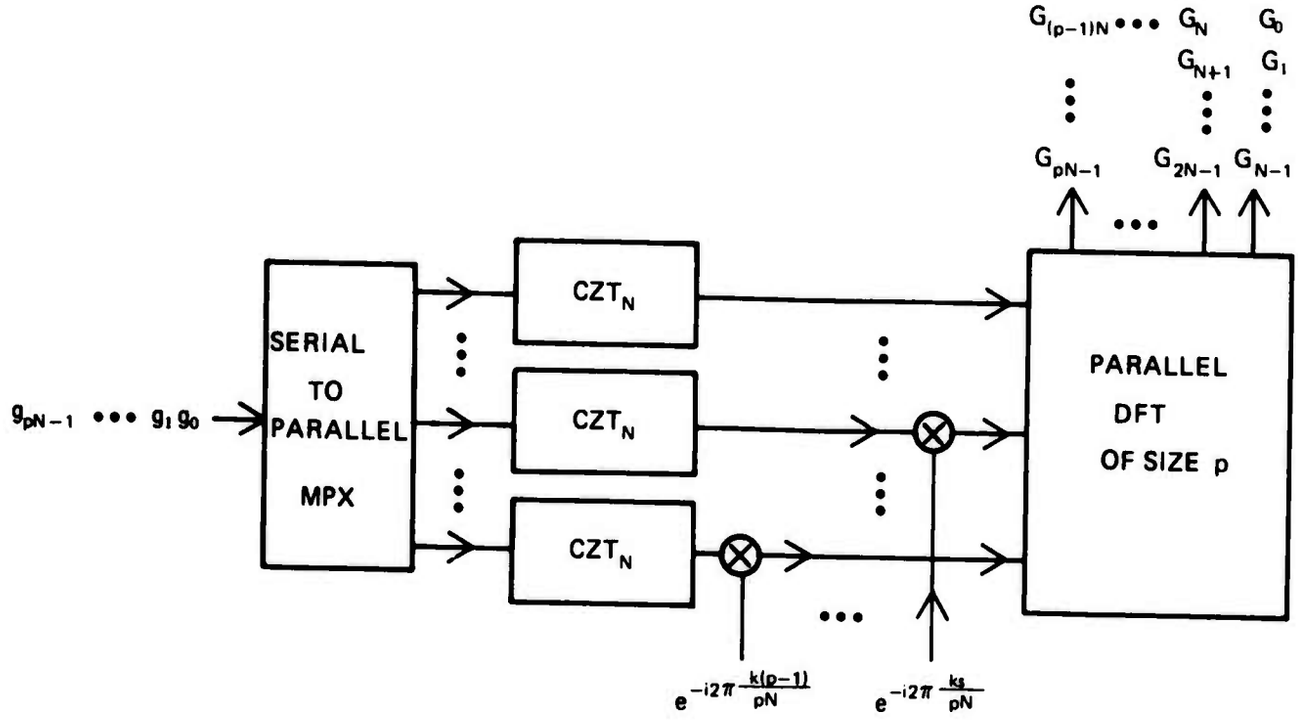


Figure 2. Organization of Modular CZT

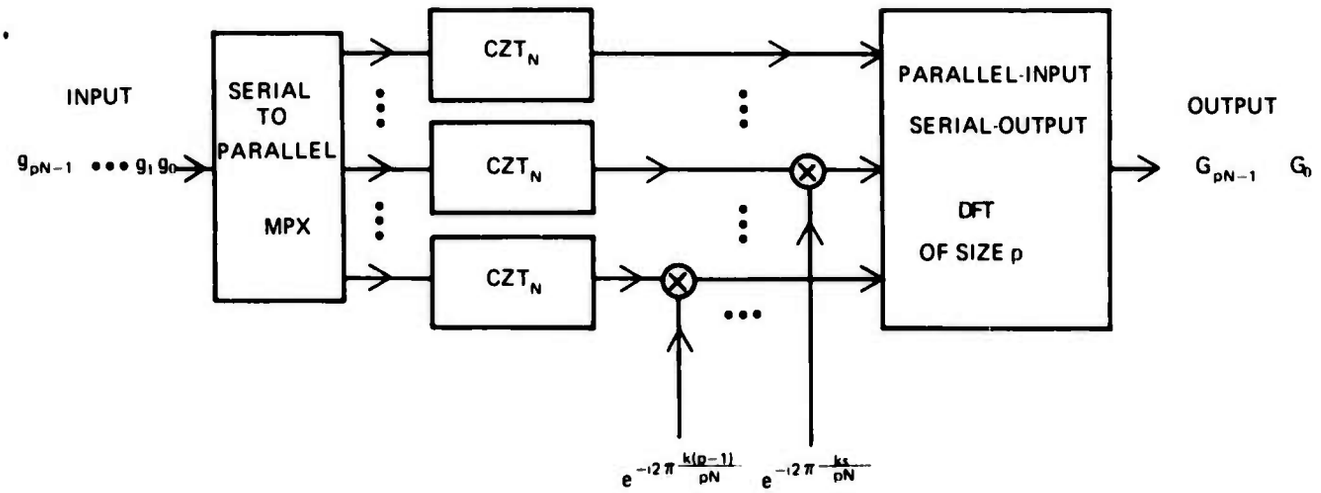


Figure 2A. Alternate Organization of Modular CZT

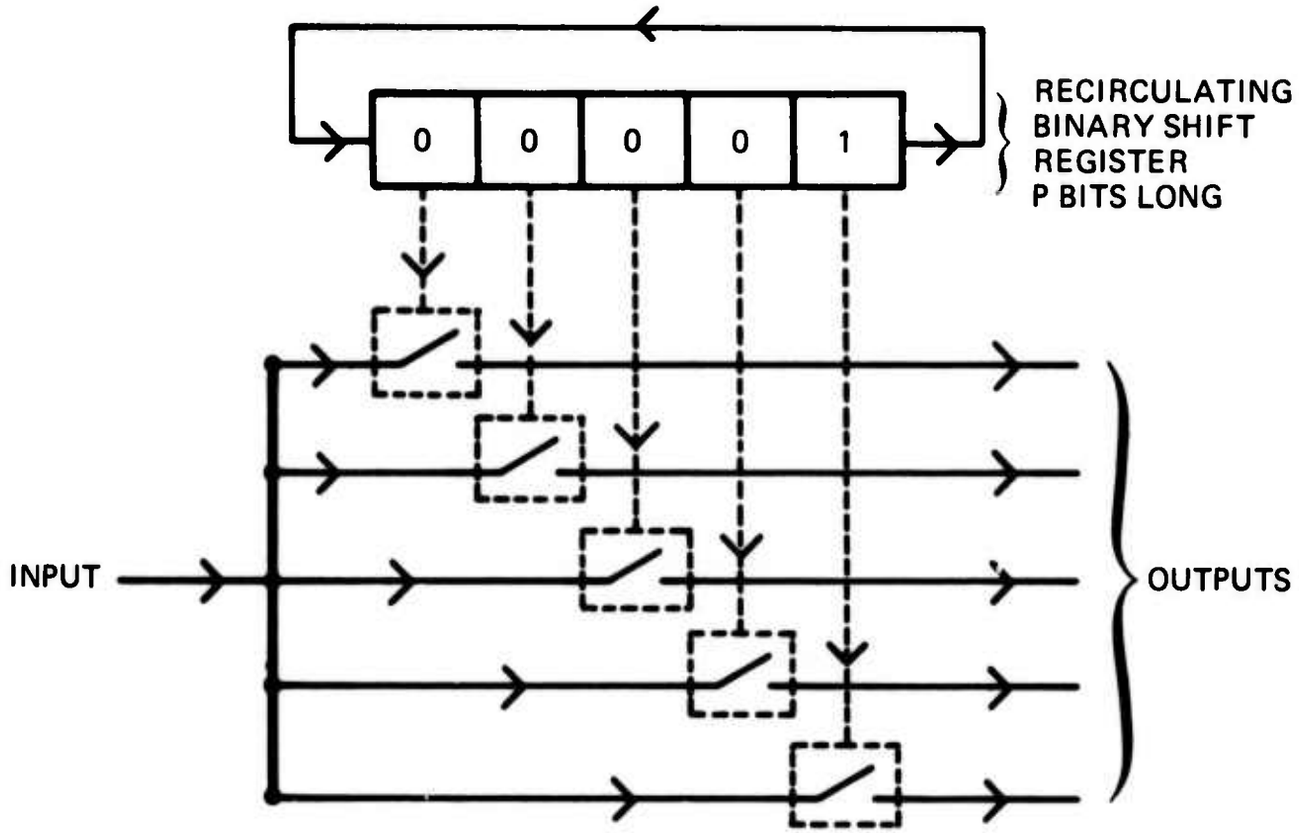


Figure 3. Serial to Parallel Multiplexer

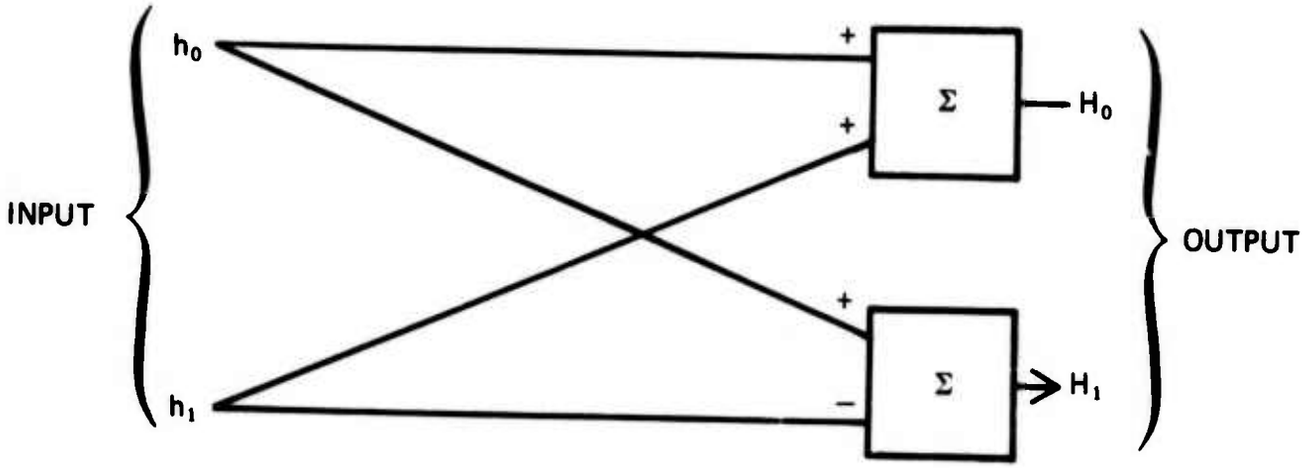


Figure 4. Parallel DFT of size P = 2

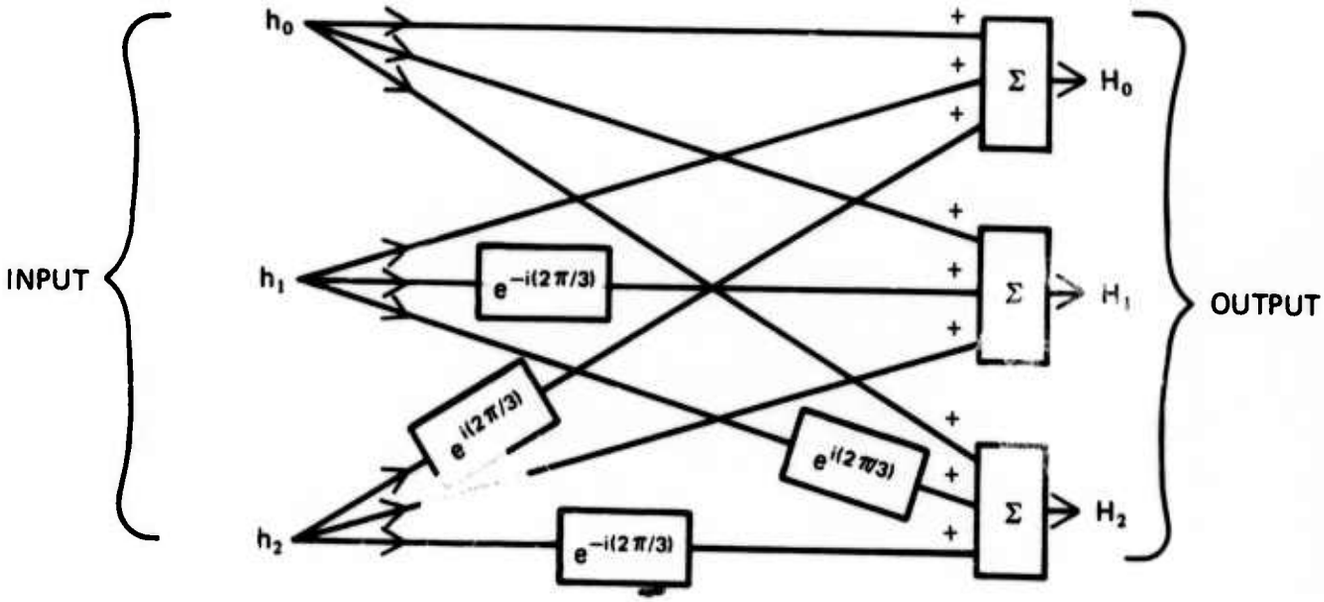


Figure 5. Parallel DFT of size P = 3

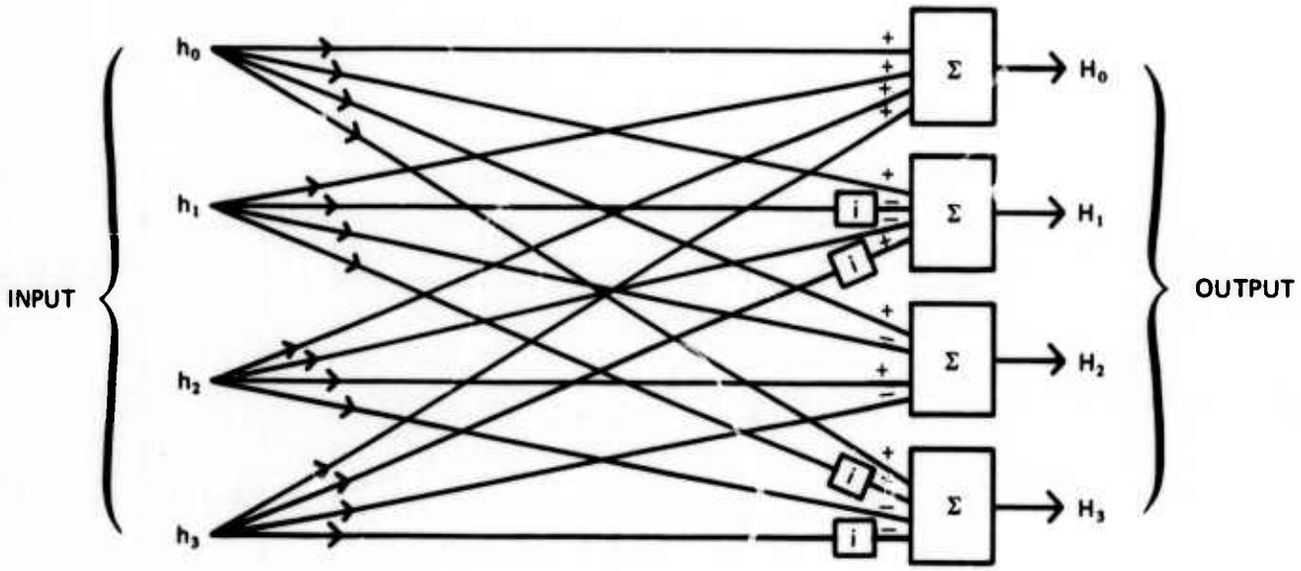


Figure 6. Parallel DFT of size P = 4

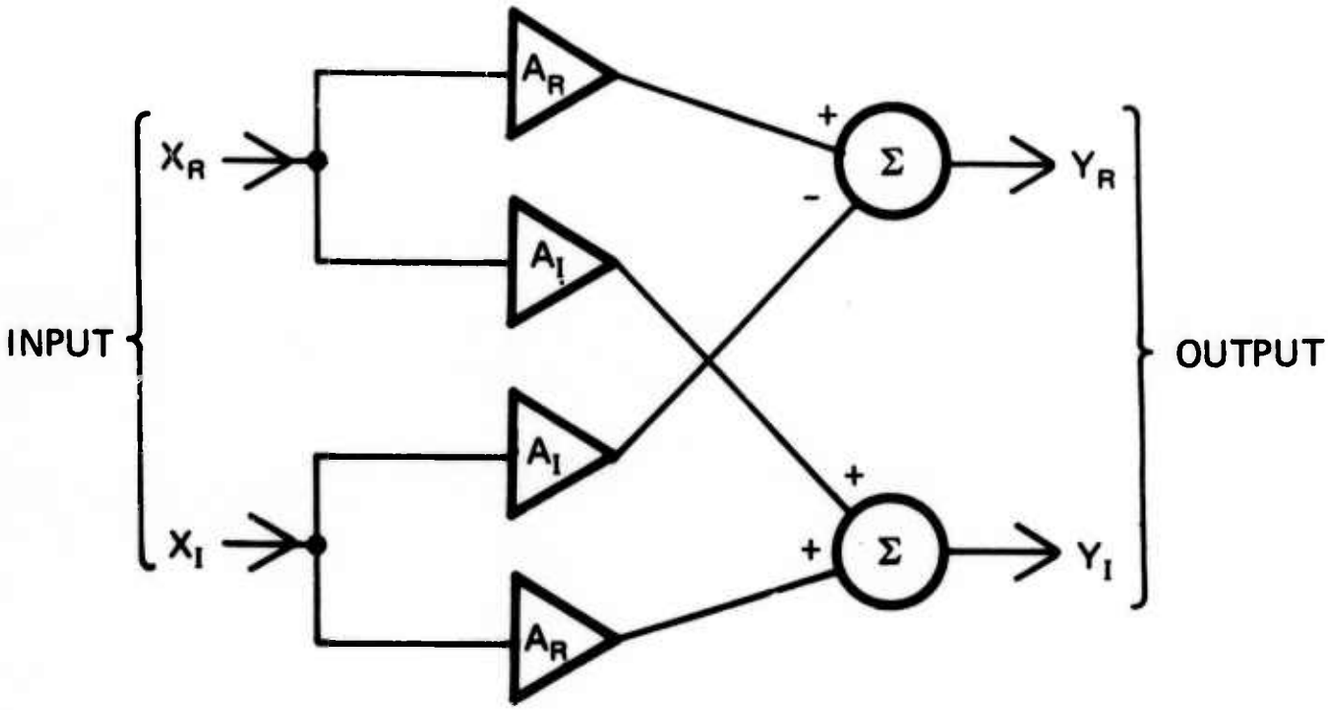


Figure 7. Complex Attenuator

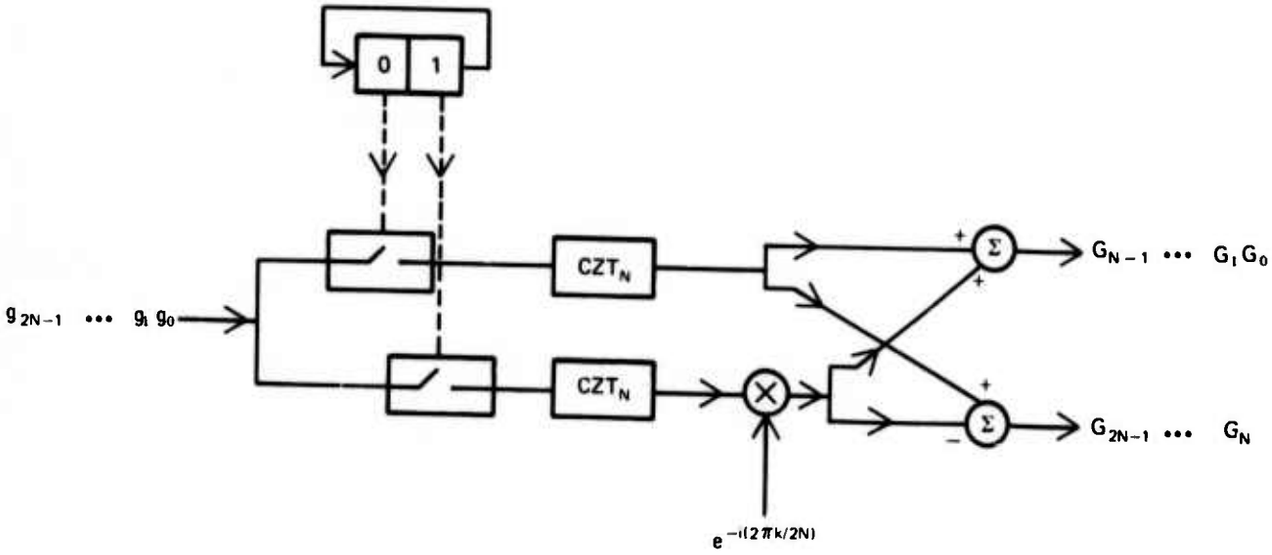


Figure 8. Double Length CZT

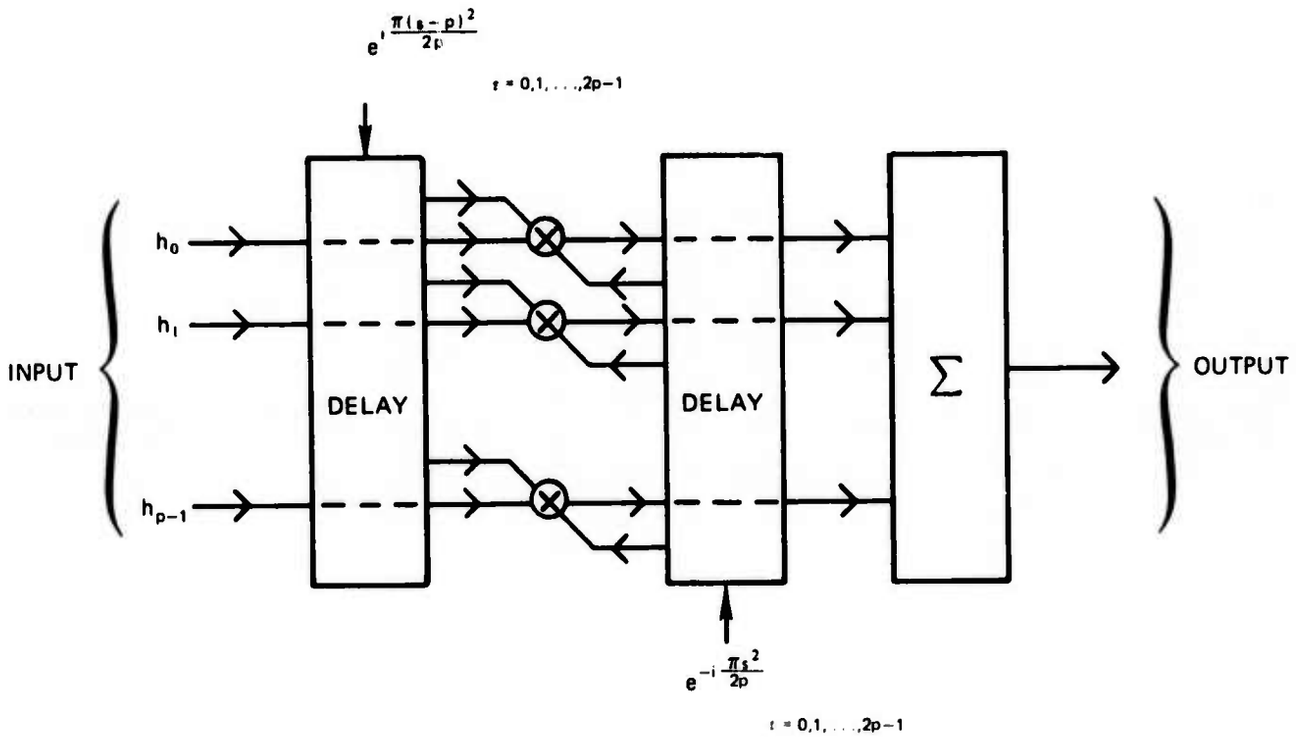


Figure 9. Parallel-Input, Serial-Output CZT using Multi-Port Transversal Filter

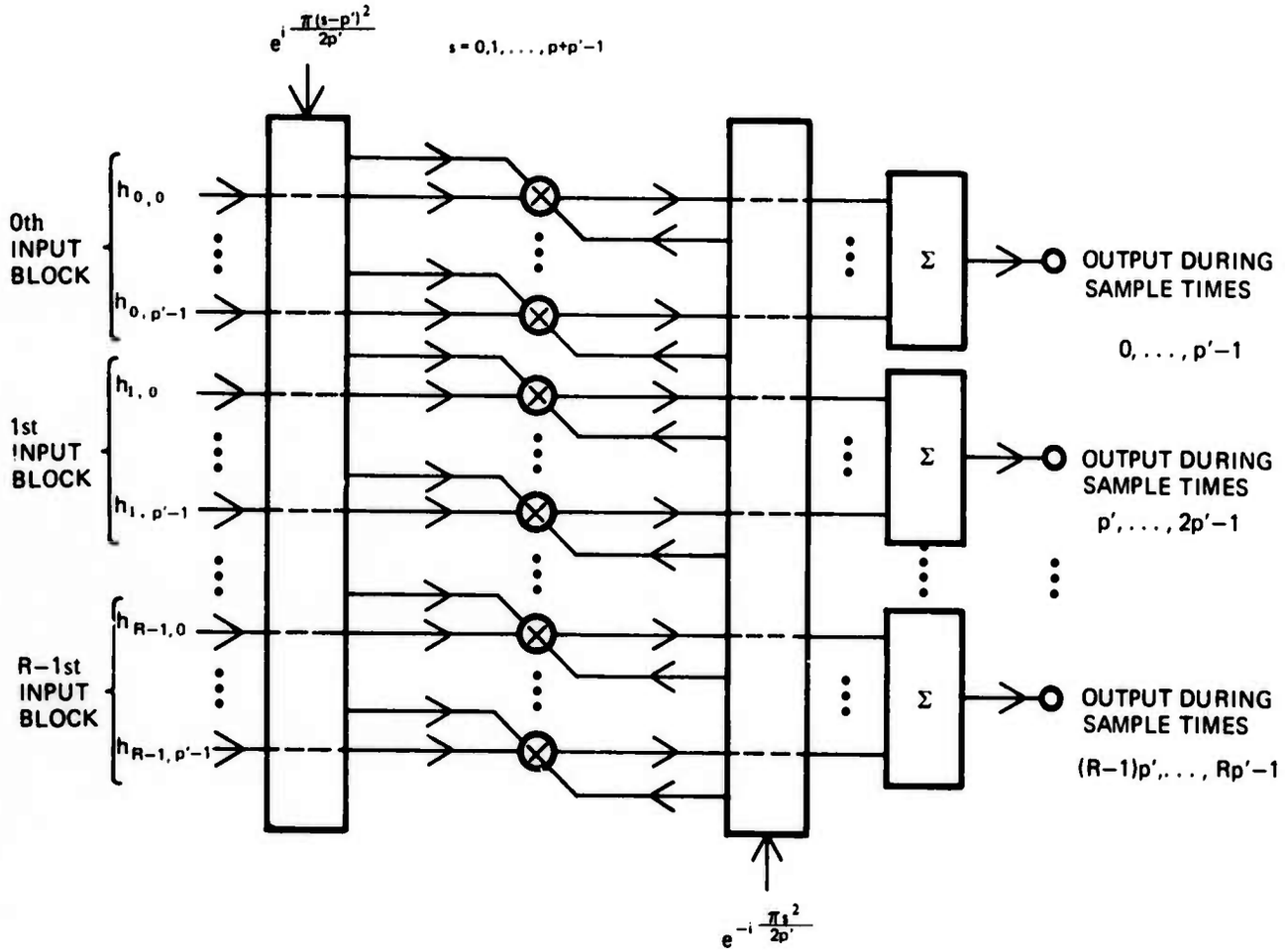


Figure 10. Modular Use of Parallel Input, Serial Output CZT Using Multiport Transversal Filter For p=Rp'

APPENDIX D

HIGH CAPACITY LINEAR PROCESSING USING MULTIDIMENSIONAL CHIRP–Z TRANSFORMS

J. M. Speiser, H. J. Whitehouse, and J. M. Alsup

Naval Undersea Center

INTRODUCTION

High speed serial access discrete Fourier transform (DFT) devices have been built using the chirp-z transform (CZT) with transversal filter implementation of the required convolution [1-3]. Such devices perform a DFT by premultiplying by a discrete chirp, convolving with a discrete chirp, and postmultiplying by a discrete chirp. One transform point is obtained per shift, and the size of the transform is proportional to the number of taps in the transversal filter used to perform the convolution. If the data is not recirculated through the filter, then 2N-1 taps are required to implement an N point DFT. Using present acoustic surface wave or CCD transversal filter technology, it is difficult to implement a transversal filter with more than about 1000 independent taps, and therefore a direct transversal filter implementation of the DFT via a one-dimensional CZT is limited to a size of about 500 points.

This note describes a DFT architecture based on a multidimensional CZT. This architecture uses a small number of high speed transversal filters together with a larger number of low speed transversal filters to implement a CZT whose speed is determined by the shift rate of the high speed transversal filters and whose transform size is determined by the total number of taps of all the low speed transversal filters.

REPRESENTATION OF A ONE-DIMENSIONAL DFT AS A TWO-DIMENSIONAL DFT

It has previously been shown [4-6] that if $N = N_1 N_2$ where N_1 and N_2 have no common divisor, then the one-dimensional DFT of equation (1) and the two-dimensional DFT of equation (2) are equivalent.

$$\hat{f}_q = \sum_{p=0}^{N-1} f_p \, e^{-i2\pi \frac{pq}{N}} \qquad \text{for } p,q = 0, 1, \ldots, N-1 \tag{1}$$

$$g(k_1,k_2) = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} g(j_1,j_2) \, e^{-i2\pi\left[\frac{j_1 k_1}{N_1} + \frac{j_2 k_2}{N_2}\right]} \tag{2}$$

$$\text{for } j_1,k_1 = 0, 1, \ldots, N_1-1$$

The scan required to make the two transforms exactly equivalent is not a simple lexical scan, but rather a linear congruential scan such as that shown in equations (3) and (4).

$$p(j_1,j_2) = j_1 N_2 + j_2 N_1 \pmod{N} \tag{3}$$

$$q(k_1,k_2) = k_1 U_1 N_2 + k_2 U_2 N_1 \pmod{N} \tag{4}$$

The constants U_1 and U_2 are the solutions of equations (5) and (6).

$$N_2 U_1 = 1 \pmod{N_1} \tag{5}$$

$$N_1 U_2 = 1 \pmod{N_2} \tag{6}$$

In special cases the output scan is very similar to the input scan. For example, if $N_2 = N_1+1$, then $U_1=1$ and $U_2=-1$, so that the two scans are given by equations (7) and (8).

$$p = j_1N_2 + j_2N_1 \qquad \text{(mod N)} \qquad (7)$$

$$q = k_1N_2 - k_2N_1 \qquad \text{(mod N)} \qquad (8)$$

SCAN CONVERSION

A linear congruential scan is illustrated in figure 1, and a means of converting the one-dimensional data to the required two-dimensional format is shown in figure 2. The multiplexer of figure 2 may be implemented with an acoustic surface wave delay line, together with auxiliary high speed analog switches. The most suitable analog switches at the present time would appear to be balanced mixers, since they are capable of switching in 1-2 nanoseconds. As shown in figures 2 and 3, the data is recirculated once. It will be noted that if N_1 and N_2 are nearly equal, the required number of taps is about $N^{.5}$, so that for a long transform the required number of taps is much smaller than the number of samples stored in the line. This greatly reduces the problem of dispersion produced by acoustic scattering from the taps. For example, only 100 taps would be needed for the multiplexer of a transform of size about 10,000. Transversal filters with 100 taps and delay lines storing 10,000 samples are well within the existing state of the art.

TWO DIMENSIONAL CZT

Once the data is in a two-dimensional format, with simultaneous serial access to all the rows, it may be transformed in the "horizontal" direction by the structure shown in figure 4. The individual one-dimensional discrete chirp filters and discrete chirp generator of figure 4 may be implemented using CCDs or digital correlators.

An acoustic surface wave device with multiple input taps may be used to access a column of the partially transformed output in a single shift time of the partial transform device. With appropriate coding of the surface wave column access device, it may also perform the discrete chirp premultiplication and discrete chirp convolution of a DFT in the "vertical" direction. A complete two-dimensional CZT architecture is shown in figure 5, and the required coding for the column access surface wave device is shown in figure 6. The complex arithmetic may be implemented as described previously [1].

A balanced mixer may be used for the fast multiplier required for the vertical transform. Lower speed variable transconductance multipliers may be used in the horizontal partial transform.

HIGH CAPACITY DFTs

The scan conversion multiplexer of figure 2 may be combined with the two-dimensional CZT of figure 5 to yield a high capacity CZT device as shown in figure 7. The throughput rate of the resulting CZT device is proportional to the sample rate of the surface wave components, and the transform size is proportional to the total number of taps of all the transversal filters or digital correlators used in the two-dimensional partial CZT.

The high capacity CZT device can be used as a high speed, high resolution spectrum analyzer, or as a high bandwidth matched filter with many degrees of freedom and variable reference function. The matched filter or cross-convolver use of the high capacity CZT device is shown in figure 8. The required column multiplexer may be realized as a small surface wave device with a single input tap and a number of unweighted output taps, together with a number of high speed analog switches, as shown in figure 9. When the device is fully loaded, the switches are all closed and the serially input data column is available in parallel at the output of the switches.

HIGH CAPACITY DFT USING A LEXICAL SCAN

The DFT device of figure 7 using the linear congruential scan of equation (3) has several limitations: the horizontal and vertical sizes of N_2 and N_1 must have no common divider, the data must recirculate through the multiplexer's acoustic surface wave device, and the output of the DFT is in the scrambled order given by equation (4).

A lexical scan similar to one half of a television interlaced scan may be used to convert a one-dimensional DFT <u>almost</u> into a two-dimensional DFT [7]. The word "almost" is used becuase of the presence of an additional phase factor which must be inserted after the first partial transform is performed. Although this scan has been previously proposed for parallel computation of the FFT [7], it lends itself equally well to the organization of parallel CZT hardware and largely circumvents the above mentioned limitations of the linear congruential scan. In particular, the lexical scan permits the complete elimination of the large surface wave delay line in the input multiplexer.

In the lexical scan defined by equations (9) and (10), $N = N_1 N_2$ may be <u>any</u> factorization of N into the product of two integers.

$$p = j_1 N_2 + j_2 \tag{9}$$

$$q = k_1 + k_2 N_1 \tag{10}$$

$$\text{for } j_1, k_1 = 0, 1, \ldots, N_1 - 1$$

$$j_2, k_2 = 0, 1, \ldots, N_2 - 1$$

The DFT of equation (1) may be rewritten in terms of the lexical scans of equations (9) and (10) as shown in equation (11) and simplified slightly in equation (12).

$$\hat{f}_{k_1+k_2N_1} = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} f_{j_1N_2+j_2} \; e^{\frac{-i2\pi}{N}\left[(j_1N_2 + j_2)\,(k_1 + k_2N_1)\right]} \tag{11}$$

$$\hat{f}_{k_1+k_2N_1} = \sum_{j_2=0}^{N_2-1} e^{-i2\pi \frac{j_2k_2}{N_2}} \; e^{-i2\pi \frac{j_2k_1}{N}} \sum_{j_1=0}^{N_1-1} f_{j_1N_2+j_2} \; e^{-i2\pi \frac{j_1k_1}{N_1}} \tag{12}$$

$$\text{for } k_1 = 0, 1, \ldots, N_1-1$$

$$k_2 = 0, 1, \ldots, N_2-1$$

The structure of a CZT architecture using equation (12) is identical to that shown in figure 7, but the scan differs from that of figure 1, and the required partial CZT differs from that of figure 4. The postmultiplier of the partial CZT for the lexical scan can not be the same for all horizontal channels because of the presence of the additional factor of $e^{-i2\pi \frac{j_2k_1}{N}}$. In essence, the postmultiplier chirps of the partial CZT using the lexical scan must be on different (discrete) carrier frequencies.

It will be noted that when a lexical scan is used, the order in which the horizontal and vertical transforms are performed is critical. For the scans shown in figure 10, the vertical transform must be performed first. Unfortunately, this prevents the transform points from coming out in natural order. However, this should not present any difficulty for performing convolution by multiplying in the frequency domain.

REFERENCES

[1] Whitehouse, H. J., J. M. Speiser, and R. W. Means, High Speed Serial Access Linear Transform Implementations, presented at the All Applications Digital Computer (AADC) Symposium, Orlando, Florida, 23-25 January 1973, reprinted as NUC TN 1026.

[2] Means, R. W., D. D. Buss, and H. J. Whitehouse, Real Time Discrete Fourier Transforms Using Charge Transfer Devices, Proceedings of the CCD Applications Conference held at the Naval Electronics Laboratory Center, San Diego, Calif., 18-20 Sept. 1973, pp. 95-101.

[3] Alsup, J. M., R. W. Means, and H. J. Whitehouse, Real-Time Discrete Fourier Transforms Using Surface Acoustic Wave Devices, Proceedings of the IEE International Specialist Seminar on Component Performance and Systems Applications of Surface Acoustic Wave Devices, held at Aviemore, Scotland, 24-28 Sept. 1973.

[4] Preisendorfer, R. W., Introduction to Fast Fourier Transforms, Visibility Laboratory, University of California, San Diego, Spring 1967.

[5] Speiser, J. M. and H. J. Whitehouse, A Two Dimensional Discrete Fourier Transform Architecture, NUC TN 1221, 17 Oct. 1973.

[6] Cooley, James W., Peter A. W. Lewis, and Peter D. Welch, Historical Notes on the Fast Fourier Transform, IEEE Transactions on Audio and Electroacoustics, Vol. AU-15, pp. 76-79, June 1967.

[7] Gold, Ben and Theodore Bially, Parallelism in Fast Fourier Transform Hardware, IEEE Transactions on Audio and Electroacoustics, Vol. AU-21, No. 1, pp. 5-16, Feb. 1973.

EXAMPLE:

LET $N_1 = 3$

$N_2 = 4$

$$g_{j_1 j_2}$$

$$\begin{array}{c} \begin{matrix} j_2=3 & j_2=2 & j_2=1 & j_2=0 \end{matrix} \\ \begin{pmatrix} g_{03} & g_{02} & g_{01} & g_{00} \\ g_{13} & g_{12} & g_{11} & g_{10} \\ g_{23} & g_{22} & g_{21} & g_{20} \end{pmatrix} \end{array} \begin{matrix} j_1=0 \\ j_1=1 \\ j_1=2 \end{matrix}$$

$$\Downarrow$$

$$\begin{pmatrix} f_9 & f_6 & f_3 & f_0 \\ f_1 & f_{10} & f_7 & f_4 \\ f_5 & f_2 & f_{11} & f_8 \end{pmatrix}$$

Fig. 1 Illustration of the desired scan for $N_1 = 3, N_2 = 4$

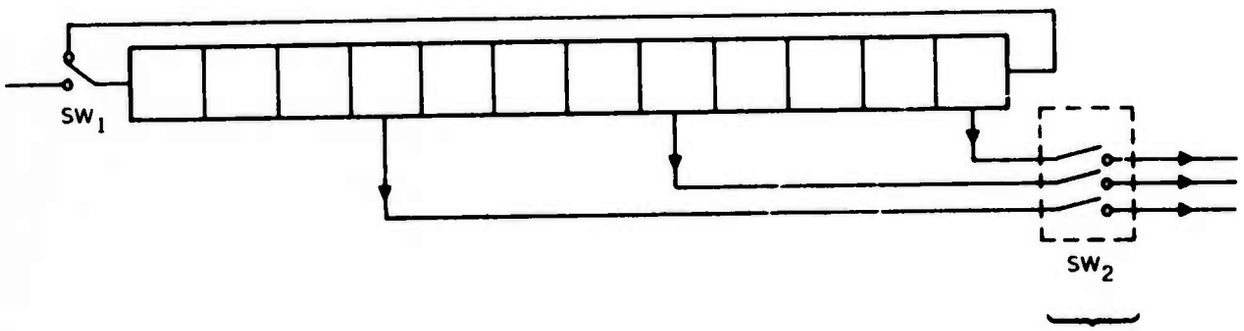


Fig. 2 Multiplexer for converting a one dimensional discrete Fourier transform into a two dimensional discrete Fourier transform (for $N_1 = 3$, $N_2 = 4$).

In general, the multiplexer will have N_1 taps, and will use a delay line capable of containing $N_1 N_2$ samples.

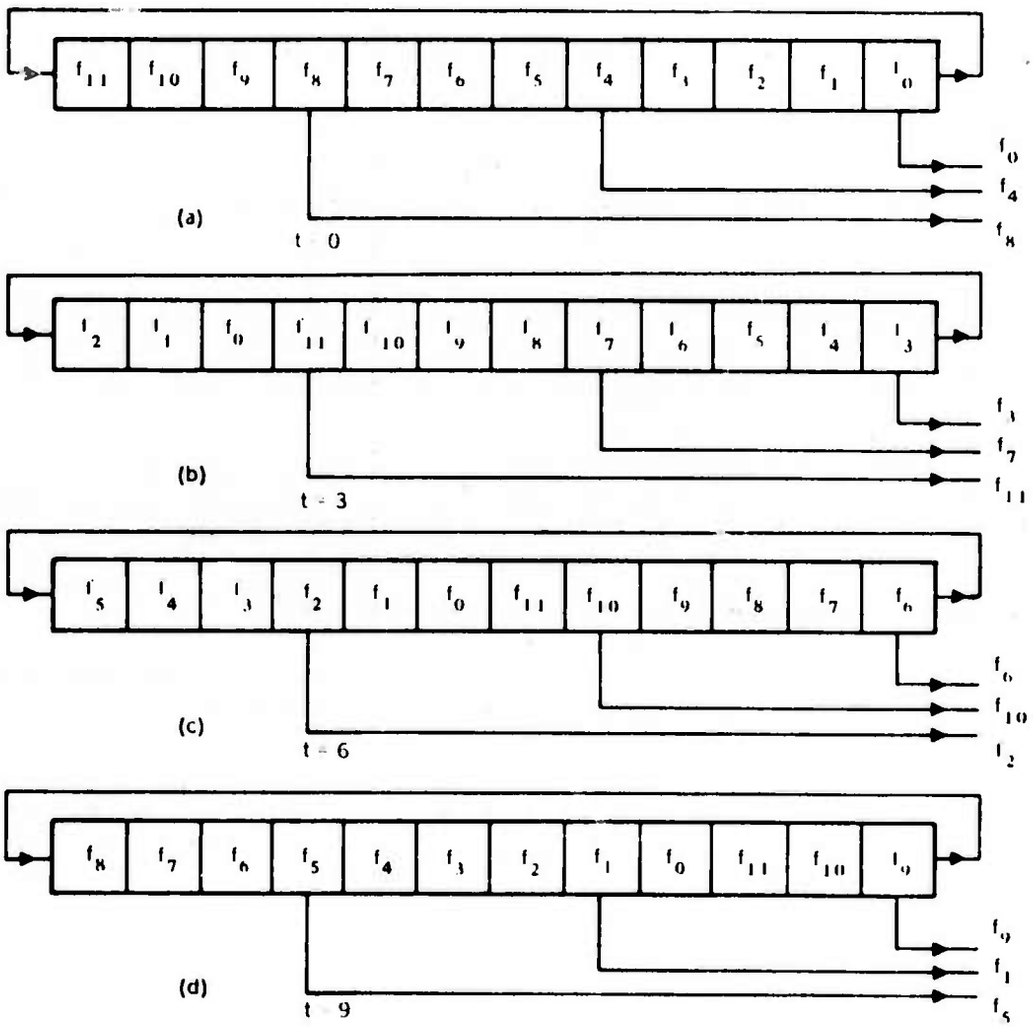


Fig. 3 Operation of the multiplexer.

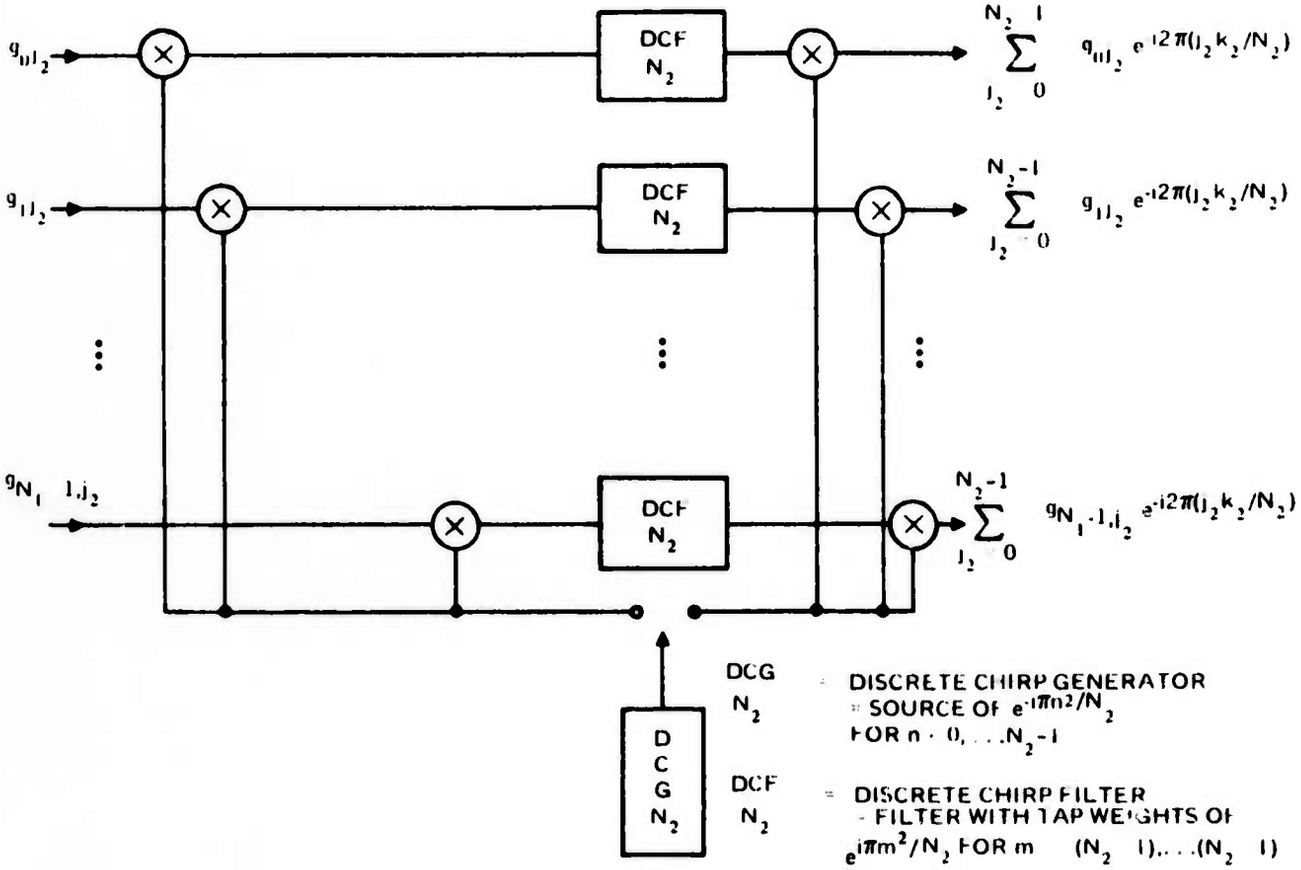


Fig. 4 Two-dimensional partial chirp-z transform.

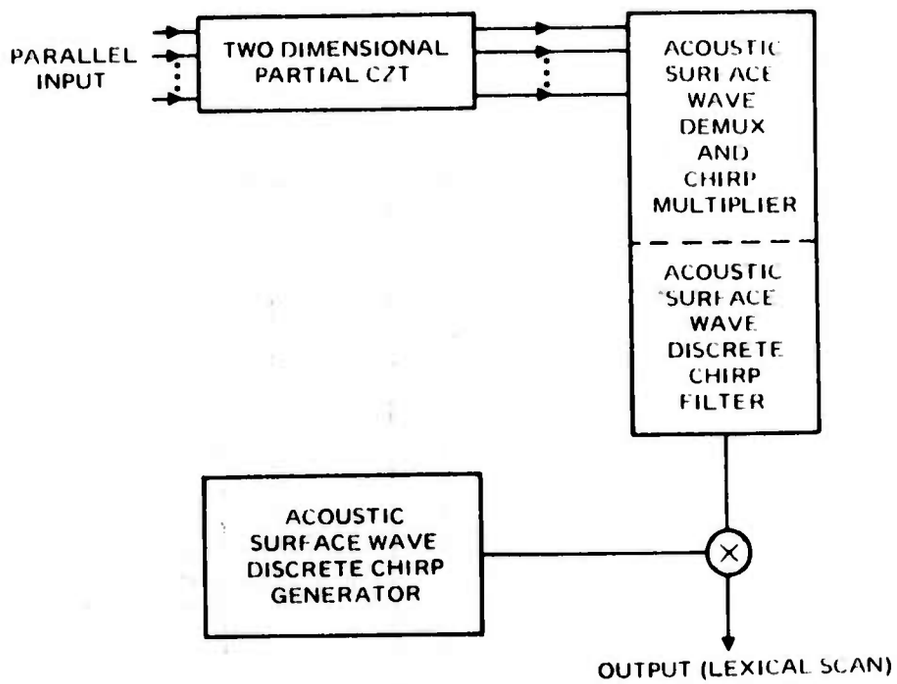


Fig. Hybrid implementation of two-dimensional CZT

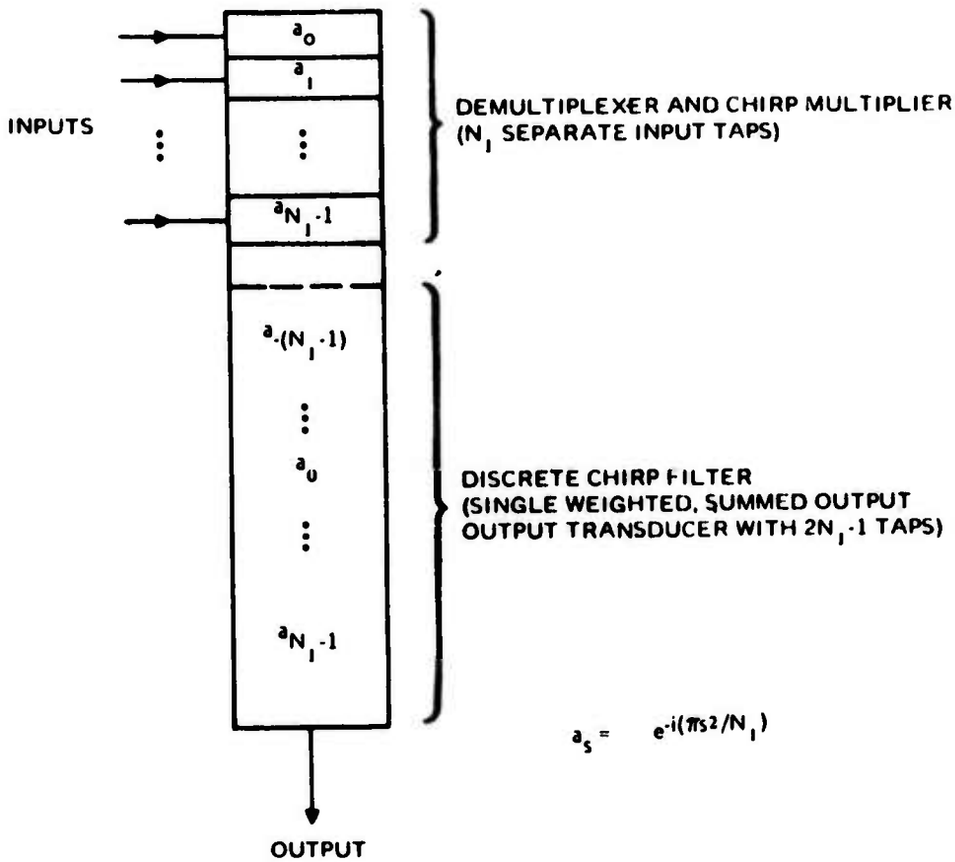


Fig. 5 Tap weights and structure for acoustic surface wave combined demultiplexer, chirp multiplier, and discrete chirp filter.

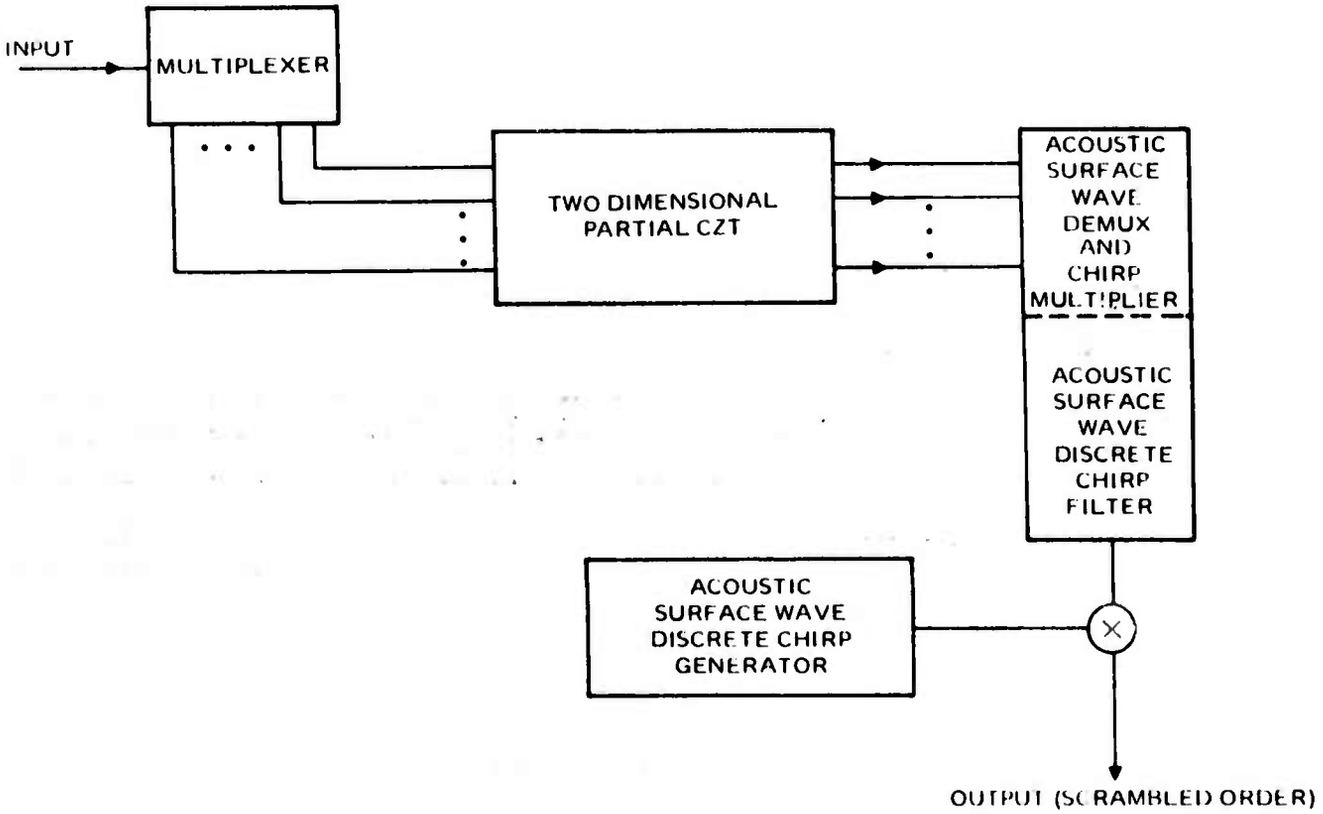


Fig. 7 Hybrid implementation of a long one-dimensional CZT.

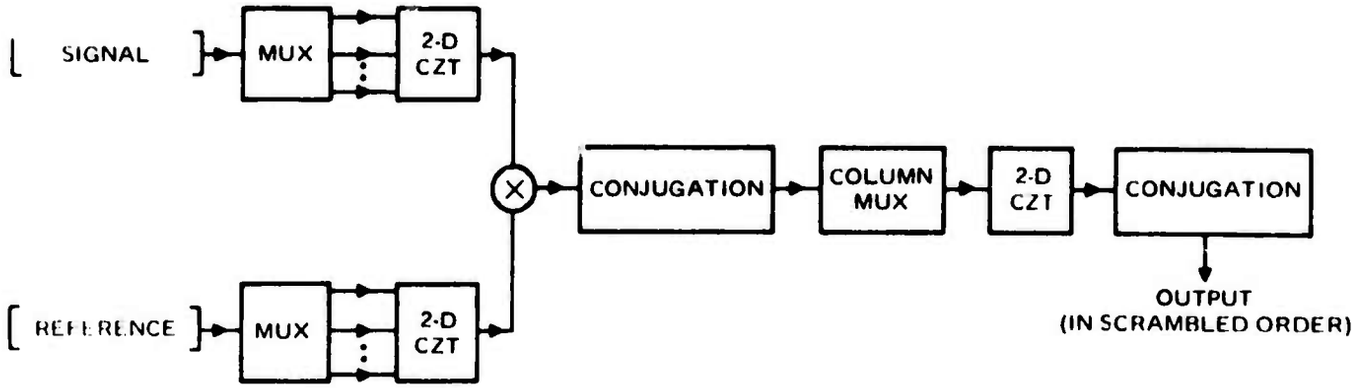


Fig. 8 Periodic convolution using 2-D CZT (output in scrambled order)

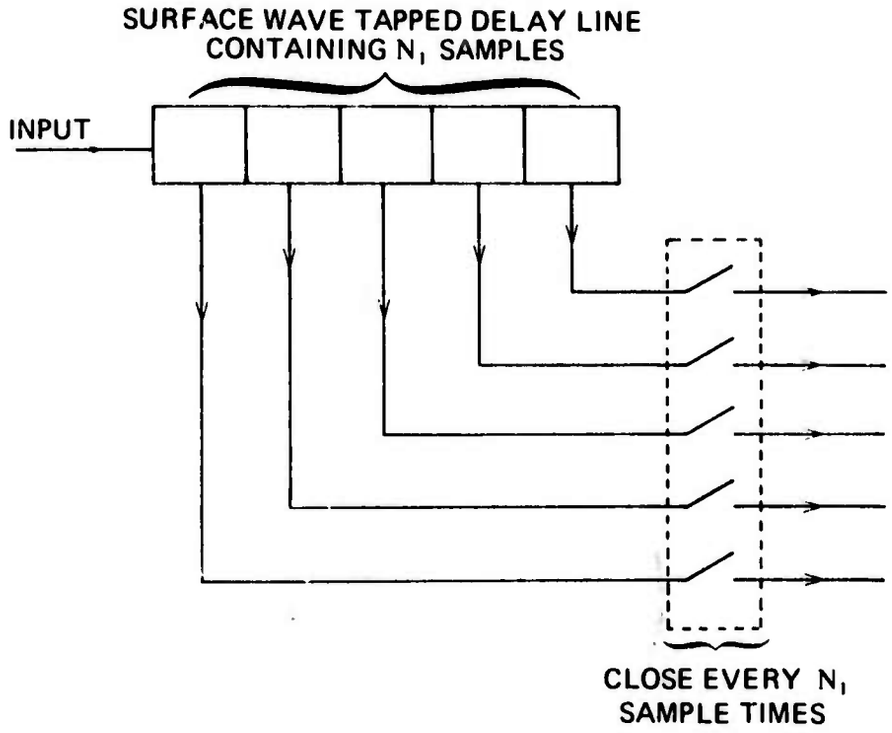


Fig. 9 Column Multiplexer

(a) $N_1 = N_2 = 3$

$$\begin{array}{ccc} j_2=2 & j_2=1 & j_2=0 \\ \begin{pmatrix} f_2 & f_1 & f_0 \\ f_5 & f_4 & f_3 \\ f_8 & f_7 & f_6 \end{pmatrix} & & \begin{array}{l} j_1=0 \\ j_1=1 \\ j_1=2 \end{array} \end{array}$$

(b) $N_1 = 3$
$N_2 = 4$

$$\begin{array}{cccc} j_2=3 & j_2=2 & j_2=1 & j_2=0 \\ \begin{pmatrix} f_3 & f_2 & f_1 & f_0 \\ f_7 & f_6 & f_5 & f_4 \\ f_{11} & f_{10} & f_9 & f_8 \end{pmatrix} & & & \begin{array}{l} j_1=0 \\ j_1=1 \\ j_1=2 \end{array} \end{array}$$

Fig. 10 Illustration of the Lexical Scan

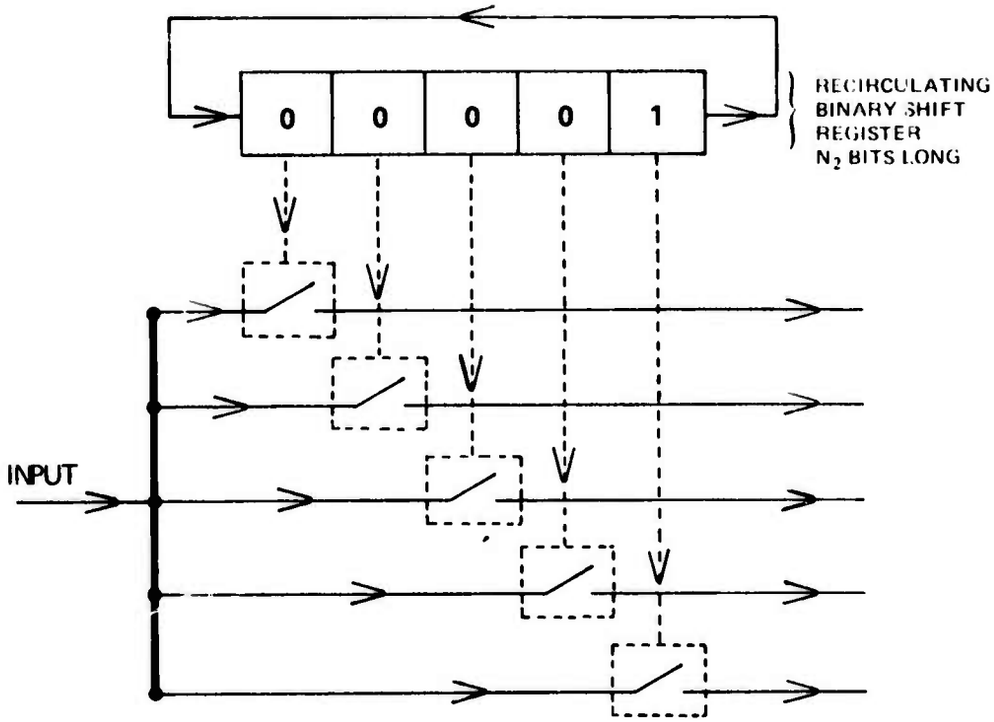


Fig. 11 Multiplexer for the Lexical Scan

APPENDIX E

STRUCTURAL ORGANIZATION FOR REAL AND COMPLEX CONVOLUTION BY IMAGING CCDs

Dr. J. W. Bond

Naval Undersea Center

STRUCTURAL ORGANIZATION FOR REAL AND COMPLEX CONVOLUTION BY IMAGING CCDs

By J. W. Bond

ABSTRACT

A mathematical model is developed for imaging CCDs with one or two registers. It is shown that an imaging CCD can be characterized by one sequence associated to the sensors and one sequence associated to each of the registers. Since only non-negative amounts of charge can be accumulated or transferred, the sequences associated with the sensors and the registers must be non-negative and real. Methods for calculating an arbitrary real or complex convolution using imaging CCDs are established by using the time domain to distinguish the real and imaginary, positive and negative terms needed to calculate the desired convolutions. Explicit architectures are described for calculating both real and complex convolutions using a CCD with either two or only one register. The recommended architectures for calculating complex convolutions are based on a generalization of Tiemann's algorithm for calculating a complex convolution by calculating the real convolution of extended sequences. More generally, any number of real and complex convolutions can be calculated interwoven in time by a device which can be used to directly calculate convolutions of sequences of non-negative real numbers.

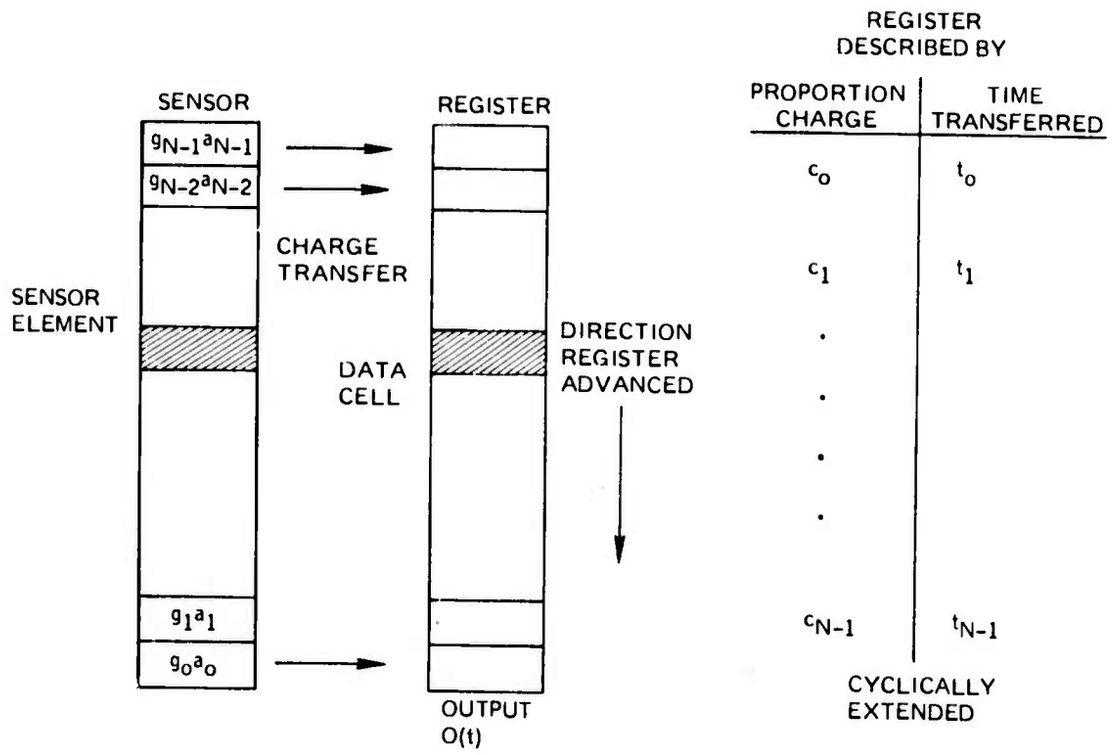
INTRODUCTION

The chirp Z algorithm reduces the calculation of a Fourier transform of a sequence of numbers to a premultiplication by a complex chirp, followed by a complex convolution by a complex chirp, followed by a post multiplication by a complex chirp. The CCD seems in principle to offer a possibility of sensing and performing the premultiplication and convolution of an image necessary to spatially Fourier transform an image. Premultiplication would be accomplished by masks over the sensors, and convolutions would be accomplished by gating the accumulated charge under the sensors to registers which perform the required delay and sum operations through charge transfer. This note grew out of a successful attempt to show that CCDs could be used to perform the Fourier Transform apart from additional

operations which do not require memory or postmultiplication. The key result described in this note is a scheme by which a charge-coupled device can be used as a quite general real convolver even though only non-negative amounts of charge can be accumulated or transferred. Once this scheme has been explained it is easy to extend usual implementations of Fourier transforms to implementations by CCDs.

The implementations described depend on an essential way on the fact that the image to be Fourier transformed, sampled by the sensors, is described by a non-negative sequence of real numbers. The signs involved in all calculations then are independent of the actual data. The image will be assumed to be varying so slowly that the image can be assumed stationary during the time required to calculate the spatial Fourier transform of the image by the CCD and its ancillary equipment.

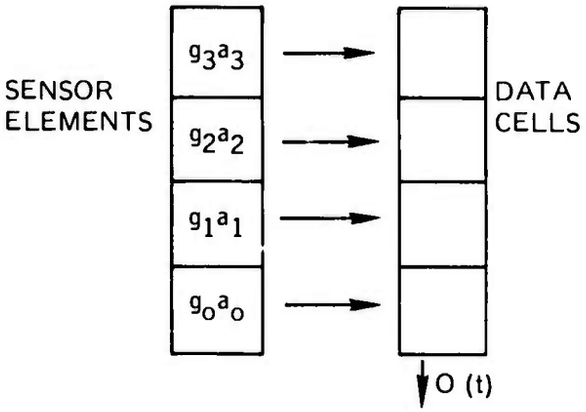
A useful mathematical description of the CCD to derive the architecture for implementing complex convolutions follows. Each CCD consists of line of sensor elements and either a right and left register each consisting of data cells or a single register with data cells. Charge accumulates under the sensors and is transferred from all sensors to some predetermined and fixed data cells in the right register or from all sensors to some predetermined and fixed data cells in the left register. After charge has been transferred to either register it is advanced from the data cell it is in to the one below it before the next charge is transferred to either register. Figure 1 illustrates the basic one register device fundamental to this note. For ease of presentation, the number of data cells per register and number of sensor elements will be taken the same so that transfers of charge in the device are always to the data cells adjacent to a sensor element. In figure 1, $g_0, g_1, \ldots, g_{N-1}$ denote the charge accumulated in the sensor elements during a fixed accumulation time T and are therefore non-negative real numbers. These real numbers can be viewed as a non-premultiplied sample of the image on the sensors. The $g_0, g_1, \ldots, g_{N-1}$ can be premultiplied by numbers between 0 and 1 by using masks to cut down the illumination hitting the sensor elements. The numbers $a_0, a_1, \ldots, a_{N-1}$ between 0 and 1 denote the proportion of illumination passed by the masks on the corresponding sensor elements. The premultiplied $g_0, g_1, \ldots, g_{N-1}$ can all be multiplied by the same number between 0 and 1 by varying the time charge is allowed to accumulate in all the sensors. The operation of the device is described by specifying the masks



$$\text{THEN } O(t_{N+k}) = \sum_{j=0}^{N-1} g_j a_j c_{k-j}$$

Figure 1. Imaging CCD as real convolver.

and the timing sequence for charge transfer. Immediately, after charge is transferred the register is advanced, i.e. the charge in any data cell is transferred to the data cell below it. The output of such a device is given by the formula in figure 1, which is derived in Corollary 1. Figure 2 illustrates how an imaging CCD with 4 elements would be used to convolve the sequence $g_oa_o, g_1a_1, g_2a_2, g_3a_3$ with the sequence c_o, c_1, c_2, c_3 with the g_j non-negative and the a_j and c_j between 0 and 1 for j = 0, 1, 2, and 3.



c_o	t_o
c_1	t_1
c_2	t_2
c_3	t_3
c_o	t_4
c_1	t_5
c_2	t_6
c_3	t_7

$$O(t_o) = g_oa_oc_o$$

$$O(t_1) = g_1a_1c_o + g_oa_oc_1$$

$$O(t_2) = g_2a_2c_o + g_1a_1c_1 + g_oa_oc_2$$

$$O(t_3) = g_3a_3c_o + g_2a_2c_1 + g_1a_1c_2 + g_oa_oc_3$$

$$O(t_4) = g_3a_3c_1 + g_2a_2c_2 + g_1a_1c_3 + g_oa_oc_o$$

$$O(t_5) = g_3a_3c_2 + g_2a_2c_3 + g_1a_1c_o + g_oa_oc_1$$

$$O(t_6) = g_3a_3c_3 + g_2a_2c_o + g_1a_1c_1 + g_oa_oc_2$$

$$O(t_7) = g_3a_3c_o + g_2a_2c_1 + g_1a_1c_2 + g_oa_oc_3$$

NOTE: AFTER FIRST THREE TRANSFERS THE IMAGING DEVICE CYCLICALLY CONVOLVES g_oa_o, g_1a_1, g_2a_2, g_3a_3 WITH c_o, c_1, c_2, c_3.

Figure 2. Imaging CCD as a real convolver of non-negative sequences.

An imaging CCD can only be used to directly calculate convolutions of two sequences of non-negative real numbers. In order to use such a device to perform complex convolutions the following mathematical problem must be solved.

How can $G = g(a+ib) * (c+id)$ be calculated using only non-negative real numbers for a, b, c, d, sequences of real numbers between −1 and 1, and g a sequence of non-negative numbers?

A straightforward mathematical solution to this problem is to write

$$g = g\left(\frac{|a|+a}{2} - \frac{|a|-a}{2} + i\left(\frac{|b|+b}{2} - \frac{|b|-b}{2}\right)\right) * \left(\frac{|c|+c}{2} - \frac{|c|-c}{2} + i\left(\frac{|d|+d}{2} - \frac{|d|-d}{2}\right)\right)$$

Note that each of the quantities $|x| \pm x$ is non-negative so that this identity calculates the complex convolution in terms of 16 convolutions of non-negative sequences. If this mathematical solution were implemented in the most straightforward manner 16 CCDs would be required each with one register with a total of 16 sensor elements associated with each g_j, $j = 0, \ldots, N-1$. It could be implemented in one CCD with 16N sensor elements and 16N data cells. Preferred implementations are based upon generalizations of a procedure discovered by Tiemann to perform complex convolutions with a single real convolver. These implementations require 10N sensor elements and 10N data cells when an imaging CCD is used with the same number of data cells and sensor elements. If a two register CCD were used then 5N sensor elements and 10N data cells are required. In the process of discovering the recommended implementations a general theory for using CCDs to perform complex convolutions was developed and will be presented in this note.

The approach is to describe how a single CCD can be used to do calculations which become increasingly complex. First it is shown how to do convolutions of sequences $g_0 a_0, \ldots, g_{N-1} a_{N-1}$ and $c_0, c_1, \ldots, c_{N-1}$ when the a_j and c_j are real numbers between −1 and 1 with a single CCD with two registers and then with one register. Next, it is shown how to do complex convolutions of sequences $g_0 (a_0 + ib_0), \ldots, g_{N-1} (a_{N-1} + ib_{N-1})$ and $c_0 + id_0, \ldots, c_{N-1} + id_{N-1}$ when a_j, b_j, c_j, and d_j are real numbers between −1 and 1.

All the implementations derived are based on the decomposition of a real number x into two non-negative real numbers x^+ and x^- defined by $x^+ = \frac{|x|+x}{2}$ and $x^- = \frac{|x|-x}{2}$.

Other implementations are possible, in fact all the formulas derived would hold for |x| replaced by any real number at least as large as |x|. In particular, when replaced by 1 the implementations become implementations using "biased arithmetic". Furthermore, the formulas developed always involve subtracting x^- terms from corresponding x^+ terms, i.e., they make use of the fundamental identity $x^+ - x^- = x$, so that replacing x^+ by $1 + x$ and x^- by 1 would lead to still other implementations involving "biased arithmetic".

Mathematically a*c and b*c can be calculated by convolving a new sequence $a_0, b_0, a_1, b_1, \ldots, a_{N-1}, b_{N-1}$ with a new sequence $c_0, o, c_1, o, \ldots, c_{N-1}, o$. This simple observation is the basis for reducing the numbers of CCDs required in implementing real and complex convolutions. A simple notation for describing these and similar sequences which can be created from given sequences follows. Let [a, b, c, . . . , d] denote the sequence $a_0, b_0, c_0, \ldots, d_0, a_1, b_1, c_1, \ldots, d_1, \ldots, a_{N-1}, b_{N-1}, c_{N-1}, \ldots, C_{N-1}, \ldots, d_{N-1}$, where a is the sequence $a_0, a_1, \ldots, a_{N-1}$, b the sequence $b_0, b_1, \ldots, b_{N-1}$, c the sequence $c_0, c_1, \ldots, c_{N-1}, \ldots$, d the sequence $d_0, d_1, \ldots, d_{N-1}$. In this notation the above observation is

$$[a, b] * [c, 0] = [a*c, b*c]$$

and Tiemann's algorithm is

$$[a, b, 0] * [c, +d, -c] = [\#, a*d + b*c, -(a*c - b*d)],$$

where # is of no interest. The recommended implementation of a complex convolution by a CCD with two registers is based on convolving

$$[a^+, b^+, a^-, b^-, a^+]$$

with

$$[c^+, d^+, 0, 0, 0] \text{ by one register, and}$$

with

$$[0, 0, 0, c^-, d^-] \text{ by the other register.}$$

The recommended implementations using one CCD with a single register is either to convolve

$$[a^+, a^-, b^+, b^-, 0, 0, 0, 0, 0, 0]$$

with

$$[c^+, 0, d^+, 0, c^-, 0, d^-, 0, c^+],$$

or

$$[a^+, 0, b^+, 0, a^-, 0, b^-, 0, a^+, 0]$$

with

$$[c^+, c^-, d^+, d^-, 0, 0, 0, 0, 0, 0].$$

We note that CCD imagers could be designed which would be well suited for performing complex convolutions. If each sensor element were connected to the first of two data cells then a one register CCD with 5N sensor elements could be used to calculate the complex convolution of two sequences of length N. If more data cells per sensor element were available and the particular data cell among those associated to each sensor element could vary from sensor to sensor then a device with 2N sensor elements and 10N data cells could be used to perform the same complex convolution.

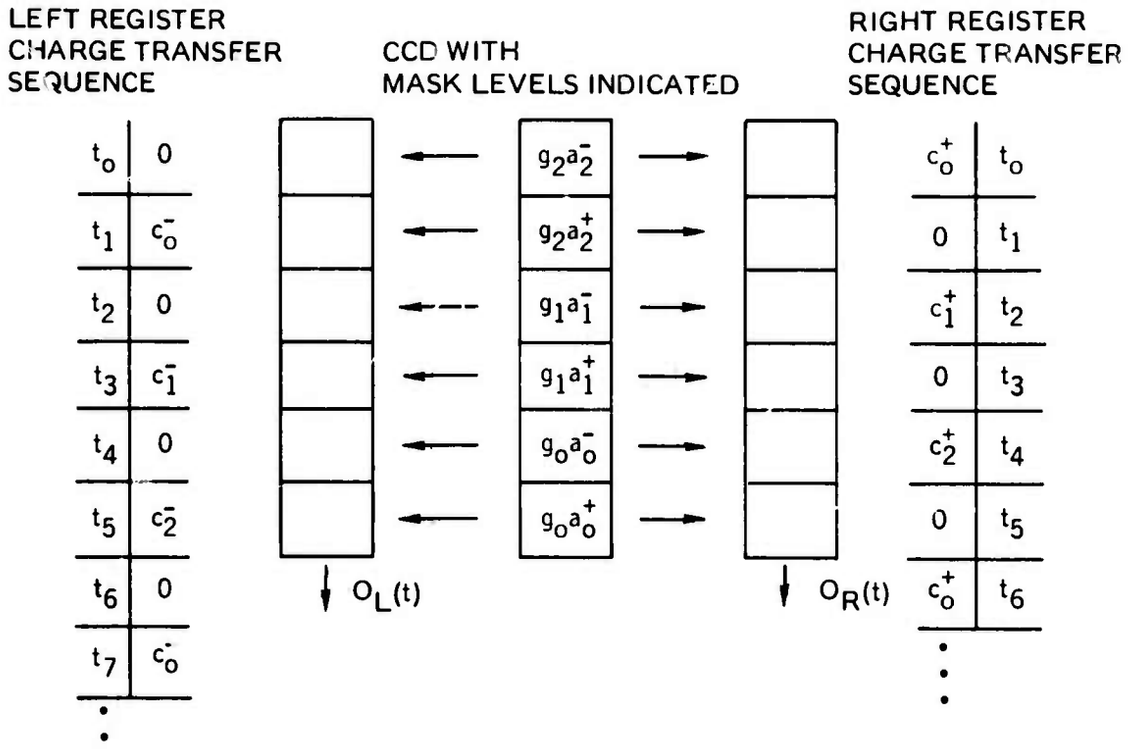
USE OF A SINGLE CCD TO PERFORM CONVOLUTIONS

In this section, recommended architectures are described for performing a real and complex convolutions using one and two register CCDs. The use of CCDs to calculate ga*c where g is a non-negative sequence of real numbers and a can c are sequences of real numbers will first be described in this section. Since convolution is linear in a and c, there is no loss of generality in restricting a can c to sequences of real numbers between -1 and 1.

Note $ga*c = g(a^+ - a^-) * (c^+ - c^-)$, where a^+, a^-, c^+, c^- are the sequences obtained from a and c by defining $a_j^+ = (|a_j| + a_j)/2$, $a_j^- = (|a_j| - a_j)/2$, $c_j^+ = (|c_j| + c_j)/2$ and $c_j^- = (|c_j| - c_j)/2$. Therefore, if the architecture is based on this identity, it is necessary to calculate four convolutions involving real sequences of non-negative numbers: $ga^+ * c^+$, $ga^- * c^+$, $ga^+ * c^-$ and $ga^- * c^-$.

If a CCD has two registers $ga^+ * c^+$ and $ga^- * c^+$ can be calculated in the right register and $ga^+ * c^-$ and $ga^- * c^-$ can be calculated in the left register. Figure 3 shows by example how one CCD with two registers can be used to calculate ga*c.

If a CCD has only one register, $ga^+ * c^+$, $ga^- * c^+$, $ga^+ * c^-$, and $ga^- * c^-$ can be calculated in two different ways as illustrated in Figures 4 and 5. Note the implementations illustrated in Figures 4 and 5 are obtained from one another by interchanging a and c. The figures in this section present the implementations for N=3. It is easy to derive the implementations for general N from those illustrated for N=3 in the figures.



$$[O_R(t_6) - O_R(t_7) = g_oa_oc_o^+ + g_1a_1c_2^+ + g_2a_2c_1^+]$$

$$- [O_L(t_7) - O_L(t_8) = g_oa_oc_o^- + g_1a_1c_2^- + g_2a_2c_1^-]$$

$$= g_oa_oc_o + g_1a_1c_2 + g_2a_2c_1$$

$$[O_R(t_8) - O_R(t_9) = g_oa_oc_1^+ + g_1a_1c_o^+ + g_2a_2c_2^+]$$

$$- [O_L(t_9) - O_R(t_{10}) = g_oa_oc_1^- + g_1a_1c_o^- + g_2a_2c_2^-]$$

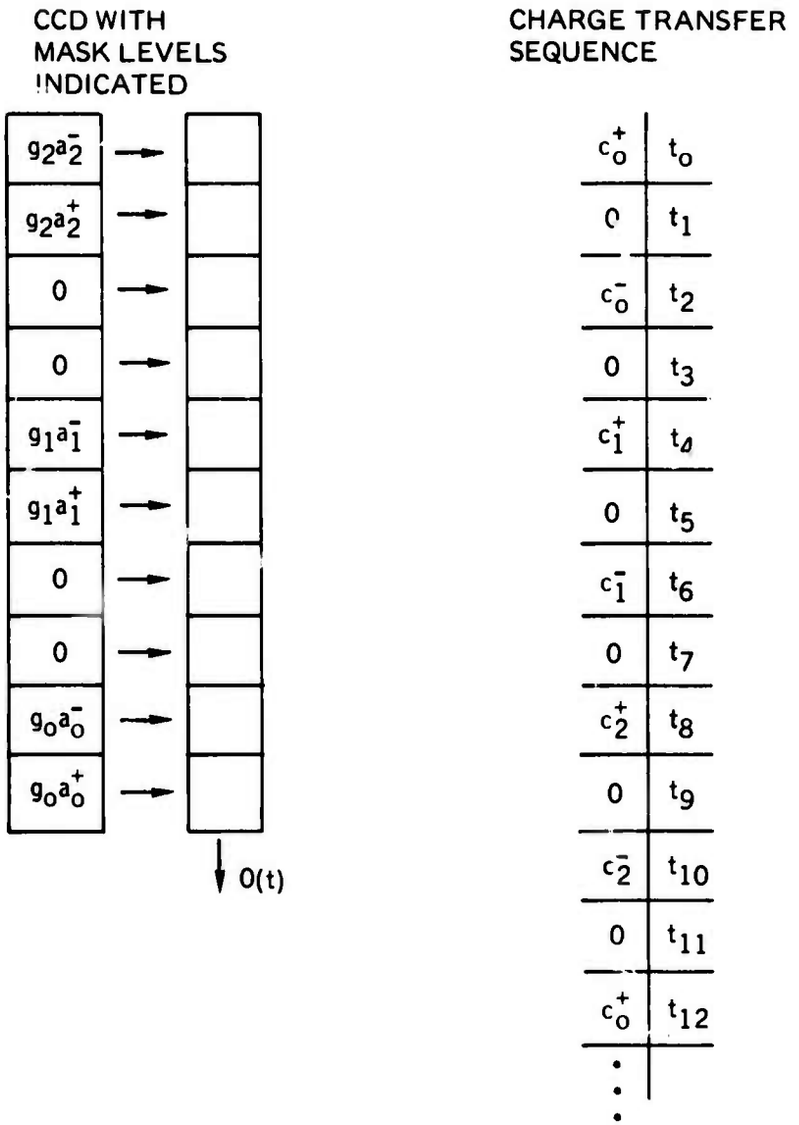
$$= g_oa_oc_1 + g_1a_1c_o + g_2a_2c_2$$

$$[O_R(t_{10}) - O_R(t_{11}) = g_oa_oc_2^+ + g_1a_1c_1^+ + g_2a_2c_o^+]$$

$$- [O_L(t_{11}) - O_R(t_{12}) = g_oa_oc_2^- + g_1a_1c_1^- + g_2a_2c_o^-]$$

$$= g_oa_oc_2 + g_1a_1c_1 + g_2a_2c_o$$

Figure 3. Real convolution of length 3 sequences with a 2 register CCD.

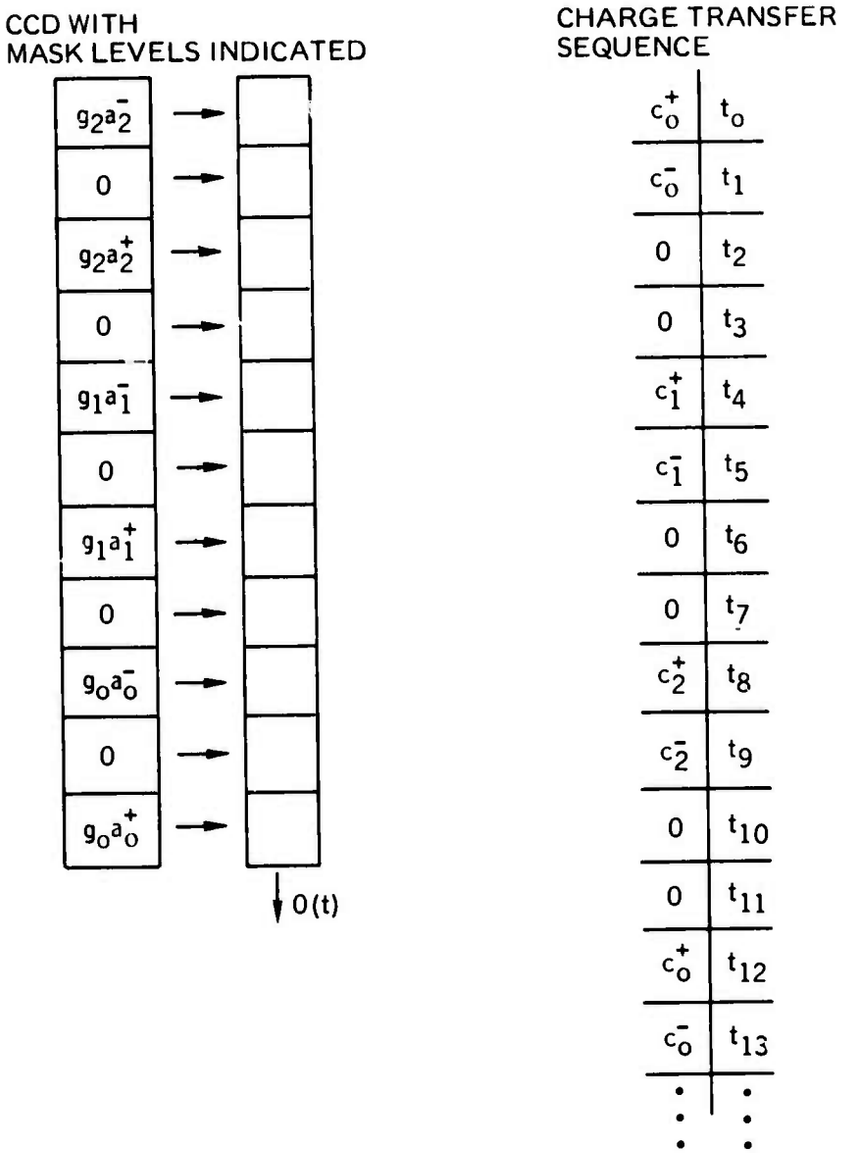


$$O(t_{12}) - O(t_{13}) - O(t_{14}) + O(t_{15}) = g_oa_oc_o + g_1a_1c_2 + g_2a_2c_1$$

$$O(t_{16}) - O(t_{17}) - O(t_{18}) + O(t_{19}) = g_oa_oc_1 + g_1a_1c_o + g_2a_2c_2$$

$$O(t_{20}) - O(t_{21}) - O(t_{22}) + O(t_{22}) = g_oa_oc_2 + g_1a_1c_1 + g_2a_2c_o$$

Figure 4. Real convolution of length 3 sequences with a one register CCD, I.



$$O(t_{12}) - O(t_{13}) - O(t_{14}) + O(t_{15}) = g_oa_oc_o + g_1a_1c_2 + g_2a_2c_1$$

$$O(t_{16}) - O(t_{17}) - O(t_{18}) + O(t_{19}) = g_oa_oc_1 + g_1a_1c_o + g_2a_2a_2$$

$$O(t_{20}) - O(t_{21}) - O(t_{22}) + O(t_{23}) = g_oa_oc_2 + g_1a_1c_1 + g_2a_2c_o$$

Figure 5. Real convolution of length 3 sequences with a one register CCD, II.

The implementations of the complex convolution g(a+ib)*(c+id) which are described next utilize Tiemann's algorithm. Note that 16 sums would be required if the basis for the calculation were the identity $g((a^+-a^-) + i(b^+-b^-)) * ((c^+-c^-) + i(d^+-d^-)) = g(a+ib) * (c+id)$. Tiemann's algorithm can be written [a, b, 0] * [c, d, -c] = [#, ad+bc, -(ac-bd)], where # denotes a term which is not useful. If implemented by a CCD in terms of a^+-a^-, b^+-b^-, c^+-c^-, d^+-d^-, 12 sums would be required. The implementations for the complex convolution described next are based on a modification of Tiemann's identity. The two register single CCD implementation is based on the convolution of $[a^+, b^+, a^-, b^-, a^+]$ with $[c^+, d^+, 0, 0, 0]$ in one register and $[c^-, d^-, 0, 0, 0]$ in the other. The one register single CCD implementations are based on either the convolution of $[a^+, a^-, b^+, b^-, 0, 0, 0, 0, 0, 0]$ with $[c^+, 0, d^+, 0, c^-, 0, d^-, 0, c^+, 0]$ or, by interchanging a and b with c and d, the convolution of $[a^+, 0, b^+, 0, a^-, 0, b^-, 0, a^+, 0]$ with $[c^+, c^-, d^+, d^-, 0, 0, 0, 0, 0, 0]$. Note these implementations only require the calculation of 10 sums, two of which are not combined with the other input used to form the desired real and imaginary parts of the output. Figure 6 illustrates the 2 register single CCD implementation and figures 7 and 8 the two different one register single CCD implementations of the complex convolution.

The implementations illustrated in figures 4 and 7 have the different masks of the same signal sample adjacent while these illustrated in figures 5 and 8 have them separated by sensor elements masked to zero. Therefore for CCDs with each sensor element corresponding to a single data cells the implementations described by figures 4 and 7 seem best because adjacent sensor elements could be located within one circle of resolution of the imager.

Note that in the CCD imagers illustrated the basic timing sequence for charge transfers can be longer than the number of sensor elements because zeros at either the beginning or end of the sequence associated to the sensors need not have corresponding sensor elements. In particular, note that the implementation of the complex convolution illustrated by figure 7 for the case N = 3 would in general require ION-6 sensor elements while that illustrated by figure 8 would require ION-1.

The difference in the total number of sensor elements required for the implementations illustrated in figures 7 and 8 is greatly exaggerated by taking N = 3 and this difference is not considered to be as important as possible design modifications which can be used to

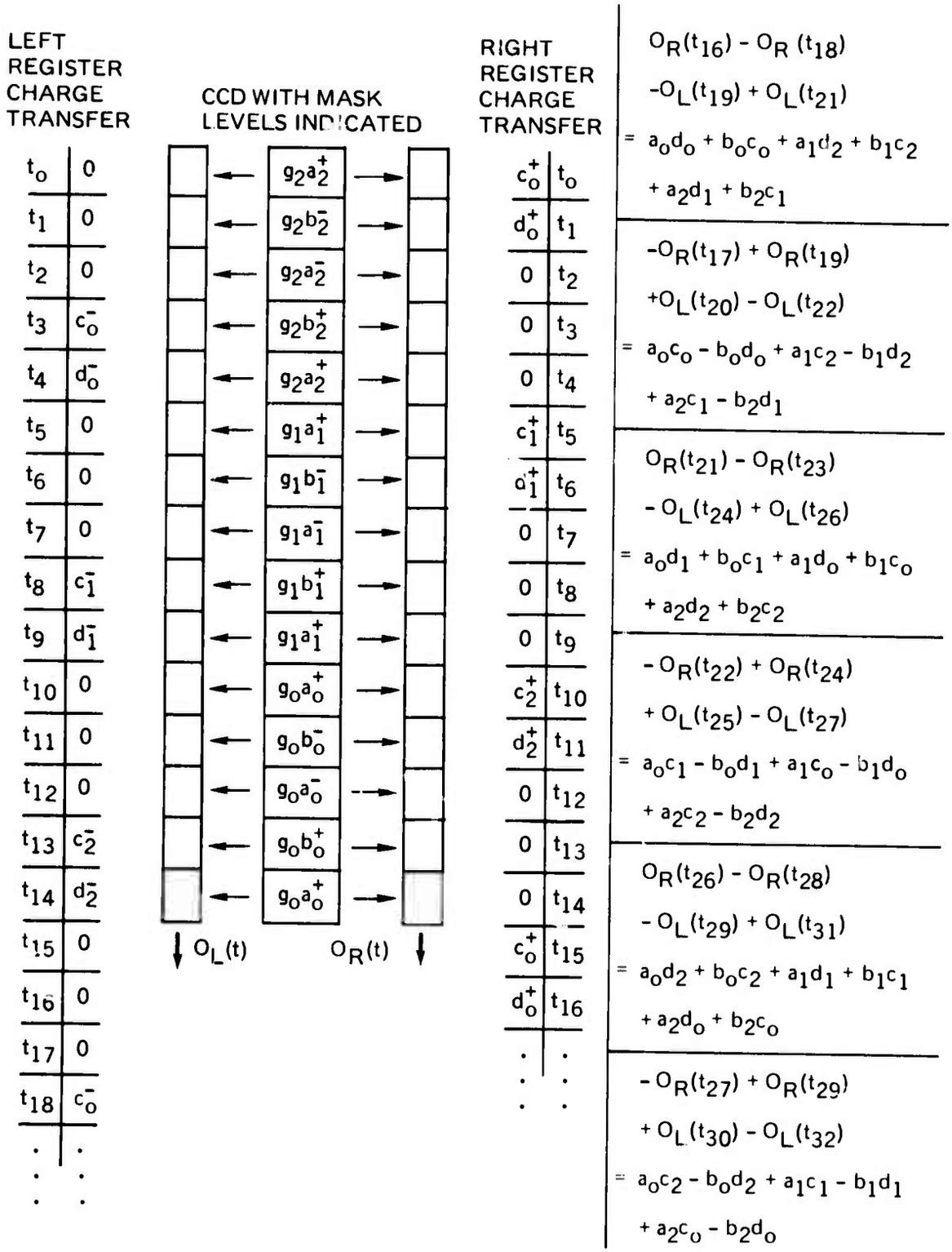
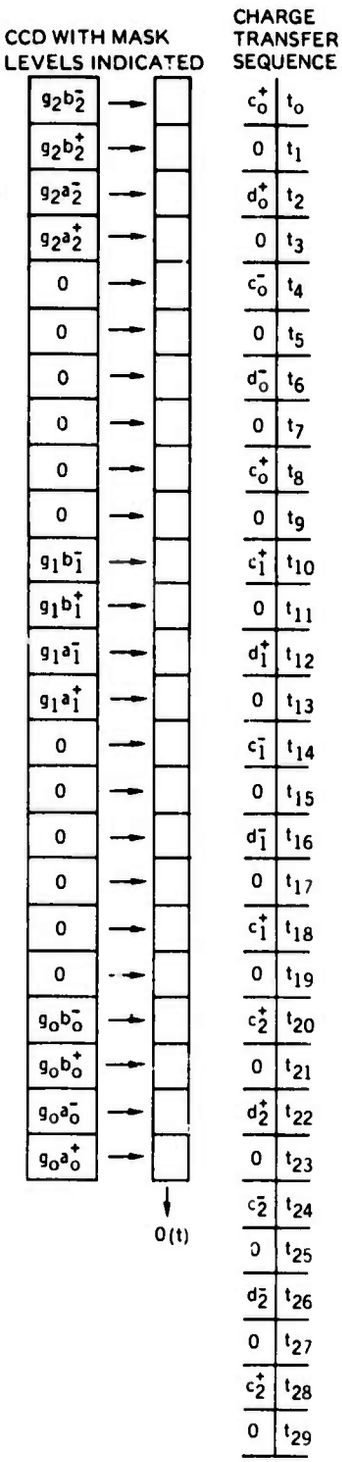


Figure 6. Complex convolution by a single 2 register CCD of sequences of length 3.



$$O(t_{32}) - O(t_{33})$$

$$- O(t_{36}) + O(t_{37})$$

$$= g_o b_o c_o + g_o a_o d_o + g_1 b_1 c_2 + g_1 a_1 d_2$$

$$+ g_2 b_2 c_1 + g_2 a_2 d_2$$

$$- O(t_{34}) + O(t_{35})$$

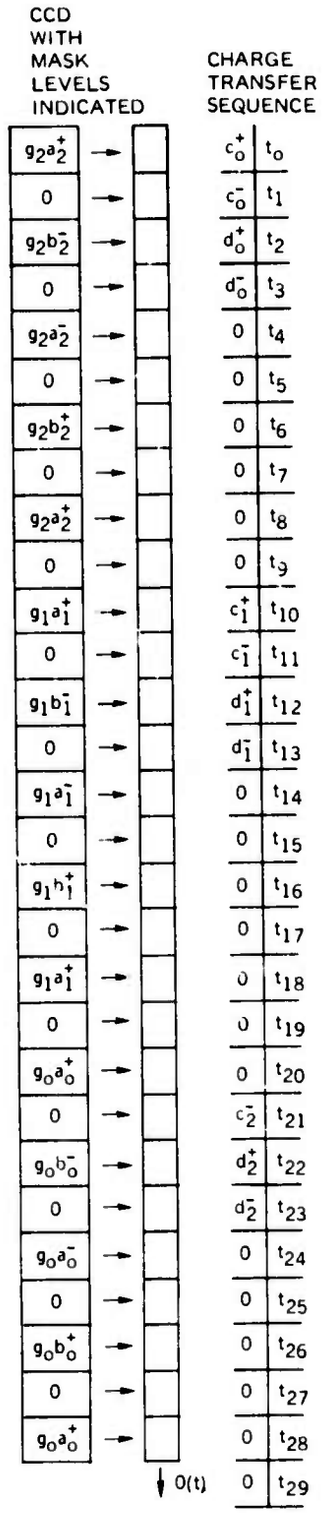
$$+ O(t_{38}) - O(t_{39})$$

$$= g_o a_o c_o - g_o b_o d_o + g_1 a_1 c_2 - g_1 b_1 d_2$$

$$+ g_2 a_2 c_1 - g_2 b_2 d_1$$

The other 4 desirable outputs are derived by calculating $O(t_{42}) - O(t_{43}) - O(t_{46}) + O(t_{47})$, $- O(t_{44}) + O(t_{45}) + O(t_{48}) - O(t_{49})$, $O(t_{52}) - O(t_{53}) - O(t_{56}) + O(t_{57})$, and $- O(t_{54}) + O(t_{55}) + O(t_{58}) - O(t_{59})$.

Figure 7. Complex convolution by a one register CCD of sequences of length 3, I.



$$O(t_{32}) - O(t_{33}) - O(t_{36}) + O(t_{37})$$

$$= g_o a_o d_o + g_o b_o c_o + g_1 a_1 d_2 + g_1 b_1 c_2 + g_2 a_2 d_1 + g_2 b_2 c_1$$

$$- O(t_{34}) + O(t_{35}) + O(t_{38}) - O(t_{39})$$

$$= g_o a_o c_o - g_o b_o d_o + g_1 a_1 c_2 - g_1 b_1 d_2 + g_2 a_2 c_1 - g_2 b_2 d_1$$

The other 4 desirable outputs are derived by calculating $O(t_{42}) - O(t_{43}) - O(t_{46}) + O(t_{47})$, $- O(t_{44}) + O(t_{45}) + O(t_{48}) - O(t_{49})$, $O(t_{52}) - O(t_{53}) - O(t_{56}) + O(t_{57})$, and $O(t_{54}) + O(t_{55}) + O(t_{58}) - O(t_{59})$.

Figure 8. Complex convolution by a one register CCD of sequences of length 3, 11.

reduce the number of sensor elements proportional to N for some implementations and not others.

Suppose that a single register CCD were designed so that each sensor element transferred its charge to the first of two data cells. Then figure 5 illustrates an implementation of the real convolution of sequences of length N which could be accomplished with a CCD of this type with 2N sensor elements and 4N data cells and figure 8 illustrates an implementation of the complex convolution of sequences of length N which could be accomplished with 5N sensor elements and 10N data cells.

Suppose the number of data cells in the CCD could differ from the number of sensor elements and furthermore that the particular data cell to which charge is transferred from a sensor element could depend on the location of the sensor element. A two register CCD designed in this manner could convolve two real sequences of length N using N sensor elements and 4N data cells. This is because for any real number x either x^+ or x^- will be zero and therefore the sign can be taken into account by the data cell to which charge is transferred. Figure 9 illustrates a scheme which would allow the implementation illustrated in figure 3 to be accomplished with only 2N sensor elements. Figure 10 provides a scheme for accomplishing the complex convolution illustrated in figure 7 using 2N sensor elements and 10N data cells with such a one register CCD.

The 2 register implementation of complex convolution by such CCDs is not so clear cut, because the Tiemann algorithm is not symmetrical in a, b and c, d. The implementations with a two register CCD illustrated in figure 6 has about 5/2 N sensors not masked to zero and uses 10N data cells. The implementation of the Tiemann algorithm based on Corollary 9 for a two register CCD has 2N sensors not masked to zero and would require 12N data cells. In both cases the scheme in figure 10 could be used to do the implementations using a minimal number of sensor elements for the given number of data cells.

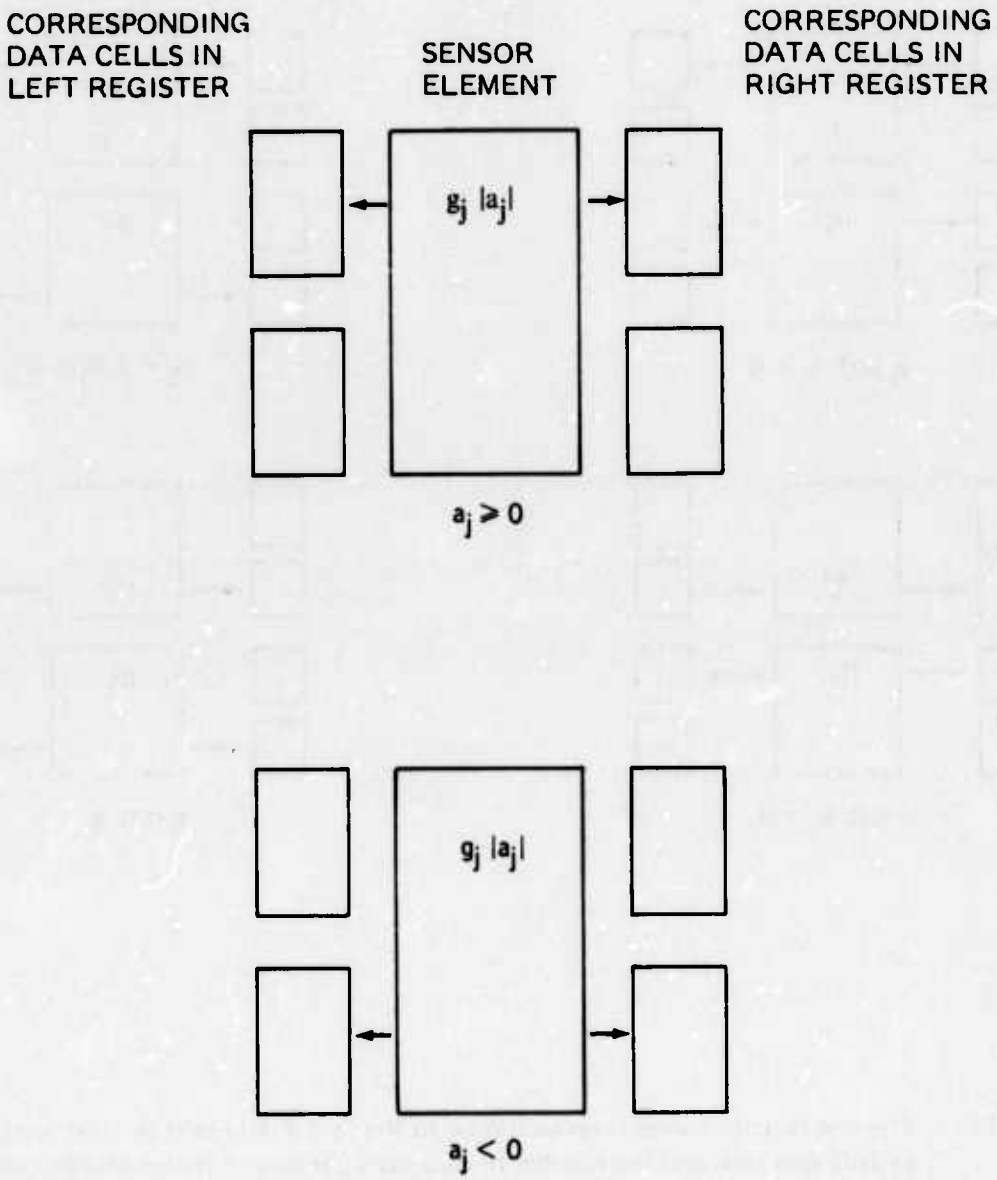
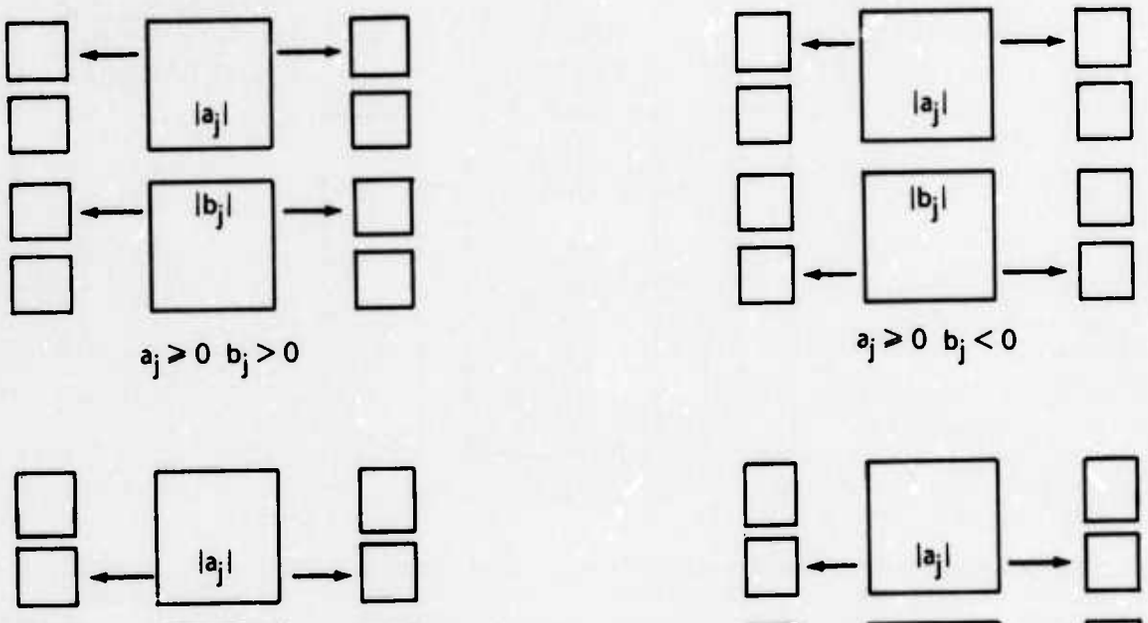


Figure 9. A scheme for performing a real convolution with a two register CCD using a minimal number of sensor elements.



NOTE: The one register connections would be to the first 4 data cells as illustrated, but to only one side, and the number of data cells per pair of sensor element would be 10.

Figure 10. Scheme for CCDs to implement complex convolution with minimal numbers of sensor elements.

1. THE MATHEMATICAL THEORY OF CALCULATION WITH ONE AND TWO REGISTER CCDs.

In this section, we develop a simple way to describe the output of a single and double register CCD as a function of time. This will enable us to derive the results discussed in the introduction and previous sections of this note in a simple manner.

Suppose a CCD consists of N sensor elements and a single register consisting of N data cells. Suppose further that $g_0, \ldots, g_{N-1}$ denotes the charge accumulated by the N sensor elements in a fixed accumulation time T. Suppose that by masking a sensor element, the amount of charge accumulated by that sensor element can be reduced. Thus masking allows us to multiply the non-negative g_j by a_j, real numbers between 0 and 1. The charge accumulated by the j-th sensor element is written $g_j a_j$; the g_j is explicitly included in some of the propositions and corollaries because in applications g_j will depend on time and a_j will not. Thus the architecture to implement a calculation will not depend on the g_j but will depend on the a_j.

To describe the output of the CCD let $c_0, c_1, \ldots, c_m, \ldots$ denote the proportion of charge which could be accumulated during fixed accumulation time T which is transferred from each sensor element to its corresponding data cell at times $t_0, t_1, \ldots, t_m, \ldots$ respectively. Without loss of generality all the c_j can be taken to be real numbers between 0 and 1. The output of the device at one of these times t_m will be denoted O(m) and is the amount of charge accumulated in the data cell corresponding to the sensor element indexed by 0. Immediately after a charge transfer the register is advanced and the charge accumulated in a data cell is transferred to the next lowest indexed data cell. The next proposition describes O(m). See figures 1 and 2.

PROPOSITION 1

The output of a CCD with N sensor elements and N corresponding data cells in its register is given by:

$$O(m) = (c_m, c_{m-1}, \ldots, c_{m-(N-1)}) \cdot (g_0 a_0, g_1 a_1, \ldots, g_{N-1} a_{N-1})$$

where

1. $g_o a_o, g_1 a_1, \ldots, g_{N-1}\, a_{N-1}$ denotes the charge accumulated in the accumulation period T
2. $c_o, c_1, c_2, \ldots, c_m, \ldots$ denotes the proportion of charge transferred from all sensor elements to corresponding data cells at times $t_o, \ldots, t_m, \ldots$
3. O(m) denotes the output of the CCD at time t_m
4. $c_j = 0$ for $j < 0$ by convention

Proof

The proof will be by induction on m and N. If N=1 the output formula is clearly true for all m. Suppose that the output formula has been established for CCD with less than N sensor elements. Suppose we have a CCD with N sensors. Then $O(0) = (c_o, 0, \ldots, 0) \cdot (g_o\, a_o, \ldots, g_{N-1}\, a_{N-1}) = g_o\, a_o\, c_o$ so that the output formula is true at time t_o. Suppose the outputs for times $t_m < t_{m+1}$ are given by

$$O(m) = (c_m, c_{m-1}, \ldots, c_{m-(N-1)}) \cdot (g_o\, a_o, \ldots, g_{N-1}\, a_{N-1}).$$

Now $O_{(m+1)} = c_{m+1}\, g_o\, a_o + C_m$, where C_m is the charge in the data cell corresponding to $g_o\, a_o$ after the charge has been transferred at time t_m and has been shifted down. C_m is therefore the output at time t_m of the CCD with N-1 sensors associated with $g_1 a_1, \ldots, g_{N-1}\, a_{N-1}$, and the same charge transfers. Therefore by induction hypothesis on N we have

$$C_m = (c_m, c_{m-1}, \ldots, c_{m-(N-2)}) \cdot (g_1\, a_1, \ldots, g_{N-1}\, a_{N-1})$$

$$= (o, c_{(m+1)-1}, c_{(m+1)-2}, \ldots, c_{(m+1)-(N-1)}) \cdot (g_o\, a_o, \ldots, g_{N-1}\, a_{N-1})$$

Therefore

$$O_{(m+1)} = (c_{m+1}, c_{(m+1)-1}, \ldots, c_{(m+1)-(N-1)}) \cdot (g_o\, a_o, \ldots, g_{N-1}\, a_{N-1})$$

which completes the proof.

Note that the sensors have been indexed from the bottom up while charge is transferred from the top down. This is necessary so that the imaging CCD can be viewed naturally as a real convolver.

Corollary 1

The output of a CCD, when $c_m = c_k$ if $m \equiv k \bmod N$ with $o \leqslant k \leqslant N-1$, is given by

$$O((N) + k) = \sum_{j=0}^{N-1} g_j a_j c_{k-j}, \text{ for } o \leqslant k \leqslant N-1 \quad .$$

Proof

$$O(N+k) = (c_{N+k}, c_{N+k-1}, \ldots, c_k) \cdot (g_o a_o, g_1 a_1, \ldots, g_{N-1} a_{N-1}) .$$

If

$$G_k = \sum_{j=0}^{N-1} g_j a_j c_{k-j}, \; k=0, \ldots, N-1$$

then G = (ga) * c. Thus Corollary 1 shows that the CCD can be viewed as a device which convolves the sequence $(g_o a_o, \ldots, g_{N-1} a_{N-1})$ with the sequence $(c_o, \ldots, c_{N-1})$. Furthermore the CCD is completely described by specifying the two sequences.

For sequences a, b, c, . . . , d all of length N let [a, b, c, . . . , d] denote the sequence

$$a_o, b_o, c_o, \ldots, d_o, a_1, b_1, c_1, \ldots, d_1, \ldots, a_{N-1}, b_{N-1}, c_{N-1}, \ldots, d_{N-1} \quad .$$

PROPOSITION 2

Let a, b, c be sequences of length N. Then

$$[a, b] * [c, o] = [a * c, b * c] \quad .$$

Proof

Let d = [a, b] so that $d_{2k} = a_k$ and $d_{2k+1} = b_k$ and f = [c, o] so that $f_{2k} = c_k$ and $f_{2k+1} = 0$. Then

$$(d * f)_{2k} = \sum_{j'=0}^{2N-1} d_{j'} f_{2k-j'} = \sum_{j=0}^{N-1} d_{2j} f_{2k-2j} = \sum_{j=0}^{N-1} a_k c_{k-j}$$

and

$$(d * f)_{2k+1} = \sum_{j'=0}^{2N-1} d_{j'} f_{(2k+1)-j'} = \sum_{j=0}^{N-1} d_{2j+1} f_{(2k+1)-(2j+1)} = \sum_{j=0}^{N} b_k c_{k-j}$$

If x is any real number let

$$x^+ = (|x| + x)/2 \text{ and } x^- = (|x| - x)/2.$$

Corollary 2

A CCD with one register can be used to convolve ga with c, with a any sequence of N real numbers between -1 and 1 and c a sequence of N non-negative numbers by summing pairs of successive outputs.

Proof

Let the CCD be the one associated with the sequences $[ga^+, ga^-]$ and [c, o] and note that $(ga^+ * c) - (ga^- * c) = g(a^+ - a^-) * c = ga*c$.

Corollary 3

A single CCD with 2 registers can be used to convolve ga with c, where a and c are sequences of real numbers between -1 and 1 by summing 4 successive outputs two from one register and two from the other.

Proof

Let the sensor elements be associated with the sequence $[ga^+, ga^-]$ and one register with the sequence $[c^+, o]$ and the other register with the sequence $[o, c^-]$. Then $(ga^+ - ga^-) * (c^+ - c^-) = ga * c$.

Note by substituting for a, b, c in Proposition 2 the sequences [c, o], [d, o], and [a, b], respectively, it follows that $[[a, b], [o, o]] * [[c, o], [d, o]] = [[a, b] * [c, o], [a, b] * [d, o]] = [a * c, a * d, b * c, b * d]$. It is easy to generalize this observation to obtain the next theorem.

Theorem 1

If a, b, c, . . . , d are m sequences of length N and e, f, . . . , g are n sequences of length N then

$$[[a, b, c, \ldots, d], [o, \ldots, o], \ldots, [o, \ldots, o]] *$$
$$[[e, o, \ldots, o], [f, o, \ldots, o], \ldots, [g, \ldots, o]]$$
$$= [a * e, a * f, \ldots, a * g, b * e, b * f, \ldots, b * g, \ldots, d * e, d * f, \ldots, d * g]$$

Proof

First we establish the theorem for the case when m is arbitrary and n=1. Let $g = [a, b, \ldots, d]$ and $h = [f, \ldots, o]$. Then $g_{mk} = a_k$, $g_{mk+1} = b_k, \ldots, g_{mk+(m-1)} = d_k$ and $h_{mk} = f_k$ with $h_{mk+j} = 0$ for $j = 1, \ldots, m-1$. Therefore

$$(g * h)_{mk+k'} = \sum_{j'=0}^{mN-1} g_{(mk+k')-j'} h_{j'}$$

$$= \sum_{j=0}^{N-1} g_{(mk+k')-mj} k_{mj}, \text{ for } h_{j'} = 0 \text{ unless } j' = mj$$

$$= \sum_{j=0}^{N-1} g_{m(k-j)+k'} h_{mj} \cdot$$

Therefore

$$(g * h)_{mk} = \sum_{j=0}^{N-1} a_{k-j} f_j, (g * h)_{mk+1} = \sum_{j=0}^{N-1} b_{k-j} f_j, \ldots, (g * k)_{mk+m-1}$$

$$= \sum_{j=0}^{n-1} d_{k-j} f_j$$

completing the first part of the proof.

Now the proof is completed by induction on n+m. Suppose the proposition has been proved for $n+m < k$. Suppose $n+m = k$ and in addition that $n \geq 2$ for the induction step has been proved already when n = 1.

Let

$$A = [a, b, c, \ldots, d], O = [o, \ldots, o], E = [e, o, \ldots, o],$$

$$F = [f, o, \ldots, o], \ldots, G = [g, \ldots, o].$$

Then

$$[a, b, c, \ldots, d], [o, \ldots, o], \ldots, [o, \ldots, o] * [[e, o, \ldots, o], [f, o, \ldots, o],$$

$$\ldots, [g, o, \ldots o]] = [A, O, \ldots, O] * [E, F, \ldots, G] = [A * E, A * F, \ldots, A * G]$$

$$= [a * e, a * f, \ldots, a * g, b * e, b * f, \ldots, b * g, \ldots, d * e, d * f, \ldots, d * g]$$

This completes the proof of the theorem.

Corollary 4

A single CCD can be used to calculate any number of real convolutions interwoven in time provided the sequences involved are non-negative sequences of real numbers bounded by 1.

Corollary 5

A single CCD with one register can be used to calculate a convolution of a sequence ga and c with a and c both sequences of real numbers between −1 and 1 by summing 4 successive outputs of the register. In particular, let the sensors be associated with the sequence $[a^+, a^-, o, o]$ and the register associated with the sequence $[c^+, o, c^-, o]$.

Proof

$$[a^+, a^-, o, o] * [c^+, o, c^-, o] = [a^+ * c^+, a^- * c^+, a^+ * c^-, a^- * c^-]$$

and

$$(a^+ * c^+) - (a^+ * c^-) - (a^- * c^+) + (a^- * c^-) = a * c.$$

Corollary 6

One CCD with two registers can be used to calculate a complex convolution $(ga + ib) * (c+id)$ in a conventional manner with a, b, c, and d real numbers between −1 and 1. In particular, let the sensors associated with one CCD be $[a^+, a^-, b^+, b^-, o, o, o, o]$ with one register $[c^+, o, o, o, d^+, o, o, o]$ and the other register $[o, o, c^-, o, o, o, d^-, o]$.

Proof

The right register calculates

$$[a^+, a^-, b^+, b^-, o, o, o, o] * [c^+, o, o, o, d^+, o, o, o]$$
$$= [a^+ * c^+, a^- * c^+, b^+ * c^+, b^- * c^+, d^+ * d^+, a^- * d^+, b^+ * d^+, b^- * d^+]$$

so that

1) $(a^+ * c^+ - a^- * c^+) - (b^+ * d^+ - b^- * d^+) = a * c^+ - b * d^+$

and

2) $(b^+ * c^+ + b^- * c^+) + (a^+ * d^+ - a^- * d^+) = b * c^+ + a * d^+.$

The left register yields the same convolutions as the right register with c^+ replaced by c^- and d^+ replaced by d^- so that

$$(a * c^+ - b * d^+) - (a * c^- - b * d^-) = a * c - b * d$$

and

$$(b * c^+ + a * d^+) - (b * c^- + a * d^-) = b * c + a * d.$$

Corollary 7

One CCD with one register can be used to calculate a complex convolution g (a+ib) * (c+id) in a conventional manner with a, b, c, and d real numbers between -1 and 1. In particular, let the sensors be associated with the sequence

$$[a^+, a^-, b^+, b^-, o, o, o, o, o, o, o, o, o, o, o, o]$$

and the register with

$$[c^+, o, o, o, c^-, o, o, o, d^+, o, o, o, d^-, o, o, o].$$

Proof

The convolution of the two sequences clearly gives the same sequences a^+, a^-, b^+, b^- convolved with c^+, c^-, d^+, d^- as used above to construct $a * c - b * d$ and $b * c + a * d$.

Corollaries 6 and 7 do not yield the best implementation of a complex convolution using a CCD, for a better one can be derived based on an algorithm found by Tiemann.

Tiemann's procedure is an immediate corollary to the next proposition.

PROPOSITION 3

Let a, b, c, d, and e be sequences of N real numbers. Then

$$[a, b, o] * [c, d, e] = [\#, a * d + b * c, a * e + b * d]$$

where the expression for # in terms of a, b, c, d, and e is not of interest.

Proof

Let f = [a, b, c] and g = [c, d, e] so that $f_{3k} = a_k$, $f_{3k+1} = b_k$, $f_{3k+2} = 0$, $g_{3k} = c_k$, $g_{3k+1} = d_k$, and $g_{3k+2} = e_k$. Then

$$(f * g)_{3k+1} = \sum_{j=0}^{N-1} f_{3j}\, g_{(3k+1)-3j} + \sum_{j=0}^{N-1} f_{3j}\, g_{(3k+1)-(3j+1)}$$

$$= \sum_{j=0}^{N-1} a_k\, d_{k-j} + \sum_{j=0}^{N-1} b_k\, c_{k-j}$$

$$(f * g)_{3k+2} = \sum_{j=0}^{N-1} f_{3j}\, g_{(3k+2)-3j} + \sum_{j=0}^{N-1} f_{3j+1}\, g_{(3k+2)-(3j+1)}$$

$$= \sum_{j=0}^{N-1} a_j\, e_{k-j} + \sum_{j=0}^{N-1} b_j\, d_{k-j}$$

Corollary 8 (Tiemann)

Let a, b, c and d be sequences of N real numbers. Then

$$[a, b, o] * [c, d, -c] = [\#, a * d + b * c, -(a * c - b * d)]$$

where the expression for # in terms of a, b, c and d is of little interest.

Proof

Let e = -c in proposition 3.

Corollary 9

One CCD with two registers can be used to implement the Tiemann algorithm for complex convolutions. In particular, let the sensors be associated with the sequence $[a^+, a^-, b^+, b^-, o, o]$ and one register with the sequence $[c^+, o, d^+, o, c^-, o]$ and the other register with the sequence $[o, c^-, o, d^-, o, c^+]$.

Proof

Let $A = [a^+, b^+, o]$, $B = [a^-, b^-, o]$, $C = [c^+, d^+, c^-]$ and $O = [o, o, o]$ in proposition 3. Then

$$[a^+, a^-, b^+, b^-, o, o] * [c^+, o, d^+, o, c^-, o]$$

$$= [A, B] * [C, O] = [A*C, B*C]$$

Note

$$A*C = [\#, a^+ * d^+ + b^+ * c^+, (a^+ * c^- + b^+ * d^+)]$$

and

$$B*C = [\#, a^- * d^+ + b^- * c^+, (a^- * c^- + b^- * d^+) \text{ by Proposition 3.}$$

Then

$$a^+ * d^+ + b^+ * c^+ - (a^- * d^+ + b^- * c^+) = a * d^+ + b * c^+$$

and

$$(a^+ * c^- + b^+ * d^+) - (a^- * c^- + b^- * d^+) = a * c^- + b * d^+ .$$

The left register outputs can be used to calculate the same expressions, but with c^+ replaced by c^-, d^+ by d^-, and c^- by c^+. Therefore

$$(a * d^+ + b * c^+) - (a * d^- + b * c^-) = a * d + b * c$$

and

$$(a * c^+ + b * d^-) - (a * c^- + b * d^+) = a * c - b * d .$$

Corollary 10

One CCD with one register can be used to implement the Tiemann algorithm for complex convolutions. In particular, let the sensors be associated with the sequence $[a^+, a^-, b^+, b^-, o, o, o, o, o, o, o, o, o]$ and the register with the sequence $[c^+, o, d^+, o, c^-, o, c^-, o, d^-, o, c^+, o]$.

Proof

$$[[a^+, b^+, o, o, o, o], [a^-, b^-, o, o, o, o]] * [[c^+, d^+, c^-, c^-, d^-, c^+] * [o, o, o, o, o, o, o]]$$

yields all the outputs used to construct $a*c - b*d$ and $a * d + b * c$ in the Corollary 9.

Note that the register sequence described in Corollary 10 repeats c^- in the middle. This suggests that length 10N sequences might be used. Towards this end the next proposition is established.

PROPOSITION 4

Let a, b, c, d, e, f, and g denote sequences of length N. Then

$[a, b, o, o, o] * [c, d, e, f, g] = [\#, a*d + b*c, a*e + b*d, a*f + b*e, a*g + b*f]$

where the expression for # is not of interest.

Proof

Let $p = [a, b, o, o, o]$ and $q = [c, d, e, f, g]$. Then $p_{5k} = a_k$, $p_{5k+1} = b_k$, $p_{5k+j} = 0$, $j = 2, 3, 4$ and $q_{5k} = c_k$, $q_{5k+1} = d_k$, $q_{5k+2} = e_k$, $q_{5k+3} = f_k$, and $q_{5k+4} = g_k$. Then

$$(p * q)_{5k+h} = \sum_{j=0}^{N-1} p_{5j}\, q_{(5k+h)-5j} + \sum_{j=0}^{N-1} p_{5j+1}\, q_{(5k+h)-(5j+1)}$$

$$= \sum_{j=0}^{N-1} p_{5j}\, q_{5(k-j)+h} + \sum_{j=0}^{N-1} p_{5j+1}\, q_{5(k-j)+(h-1)}, \quad h = 0, 1, 2, 3, \text{ and } 4.$$

Thus $(p * q)_{5k+h}$ gives the desired terms for h = 1, 2, 3, and 4.

Corollary 11

One CCD with two registers can be used to implement a modifed Tiemann algorithm for complex convolutions. In particular, let the sensors be associated with $[a^+, b^+, a^-, b^-, a^+]$ and one register with $[c^+, d^+, 0, 0, 0]$ and the other with $[c^-, d^-, 0, 0, 0]$.

Proof

$[a^+, b^+, a^-, b^-, a^+] * [c^+, d^+, 0, 0, 0] = [\#, b^+ * c^+ + a^+ * d^+, a^- * c^+ + b^+ * d^+, b^- * c^+ + a^- * d^+, a^+ * c^+ + b^- * d^+]$. Then $(b^+*c^+ + a^+*d^+) - (b^-*c^+ + a^-*d^+) = b*c^+ + a*d^+$ $(a^+*c^+ + b^-*d^+) - (a^-*c^+ + b^+*d^+) = a*c^+ - b*d^+$. The output of other register can be used to obtain the same expressions with c^+ replaced by c^- and d^+ replaced by d^-.

Corollary 12

One CCD with one register can be used to implement a modified Tiemann algorithm for complex convolutions. In particular, let the sensors we associated with $[a^+, a^-, b^+, b^-, o, o, o, o, o, o]$ and the register with the sequence $[c^+, o, d^+, o, c^-, o, d^-, o, c^+, o]$.

Proof

Let $A = [a^+, b^+, o, o, o]$, $B = [a^-, b^-, o, o, o]$, $C = [c^+, d^+, c^-, d^-, c^+]$ and $O = [o, o, o, o, o]$. Then

$$[A, B] * [C, O] = [A*C, B*C].$$

As in the proof of Corollary 9, the expressions $a*d^+ + b*c^+$, $a*c^- + b*d^+$, $a*d^- + b*c^-$, and $a*c^+ + b*d^+$ are obtained by taking the difference of successive terms. Then

$$(a*d^+ + b*c^+) - (a*d^- + b*c^-) = a*d + b*c \text{ and } (a*c^+ + b*d^-) - (a*c^- + b*d^+) = a*c - b*d .$$

Because of the importance of Corollary 12 it is desirable to have a direct proof of it independent of most of the theoretical development presented in this section.

Direct Proof of Corollary 12

Let

$$e = [a^+, a^-, b^+, b^-, o, o, o, o, o, o]$$

$$f = [c^+, o, d^+, o, c^-, o, d^-, o, c^+, o]$$

Then

$$e_{10k} = a_k^+, e_{10k+1} = a_k^-, e_{10k+2} = b_k^+, e_{10k+3} = b_k^-, e_{10k+j} = 0 \text{ for } j = 4, \ldots, 9$$

and

$$f_{10k} = c_k^+, f_{10k+2} = d_k^+, f_{10k+4} = c_k^-, f_{10k+6} = d_k^-, f_{10k+8} = c_k^+ \text{ and } f_{10k+j} = 0 \text{ when } j = 1, 3, 5, 7, \text{ and } 9.$$

The proof will follow by calculating the 8 useful terms of $e * f$.

First

$$(e*f)_{10k+h} = \sum_{j=0}^{N-1} e_{10j}\, f_{(10k+h)-10j} + \sum_{j=0}^{N-1} e_{10j+1}\, f_{(10k+h)-(10j+1)}$$

$$+ \sum_{j=0}^{N-1} e_{10j+2}\, f_{(10k+h)-(10j+2)} + \sum_{j=0}^{N-1} e_{10j+3}\, f_{(10k+h)-(10j+3)}$$

because $e_{10k+j'} = 0$ for $k' = 4, \ldots, 9$. Next

$$(e*f)_{10k+2h} = \sum_{j=0}^{N-1} e_{10j} f_{10(k-j)+2h} + \sum_{j=0}^{N-1} e_{10j+2} f_{10(k-j)+2(h-1)}$$

and

$$(e*f)_{10k+(2h+1)} = \sum_{j=0}^{N-1} e_{10j+1} f_{10(k-j)+2h} + \sum_{j=0}^{N-1} e_{10j+3} f_{10(k-j)+2(h-1)}$$

because $f_{10k+k'} = 0$ when k' is odd.

Therefore

$$g_k = (e*f)_{10k+2h} - (e*f)_{10k+(2h+1)} = \sum_{j=0}^{N-1} (e_{10j} - e_{10j+1}) f_{10(k-j)+2h}$$

$$+ \sum_{j=0}^{N-1} (e_{10j+2} - e_{10j+3}) f_{10(k-j)+2(h-1)} .$$

Then

$$g_1 = \sum_{j=0}^{N-1} a_j d^+_{k-j} + \sum_{j=0}^{N-1} b_j c^+_{k-j}$$

$$g_2 = \sum_{j=0}^{N-1} a_j c^-_{k-j} + \sum_{j=0}^{N-1} b_j d^+_{k-j}$$

$$g_3 = \sum_{j=0}^{N-1} a_j d^-_{k-j} + \sum_{j=0}^{N-1} b_j c^-_{k-j}$$

$$g_4 = \sum_{j=0}^{N-1} a_j c^+_{k-j} + \sum_{j=0}^{N-1} b_j d^-_{k-j}$$

so that finally

$$g_1 - g_3 = \sum_{j=0}^{N-1} a_j d_{k-j} + \sum_{j=0}^{N-1} b_j c_{k-j}$$

and

$$g_4 - g_2 = \sum_{j=0}^{N-1} a_j c_{k-j} - \sum_{j=0}^{N-1} b_j d_{k-j} .$$

APPENDIX F

SIGNAL PROCESSING INTERPRETER

B. R. Woods

Naval Undersea Center

ABSTRACT

This note describes additions and modifications to a signal processing interpreter which was developed as part of the ARPA sponsored program for image transmission via spread spectrum links. The signal processing interpreter, SPIN3, is an interactive program for use at a time-sharing demand terminal. It provides the user with the equivalent of a calculator designed to perform signal processing operations and provides the software equivalent of a large number of modules for breadboarding a complete signal processing system.

This interpreter is meant to be used by engineers and scientists who are familiar with signal processing, but who may have no knowledge of programming.

The signal processing interpreter is particularly useful for the rapid investigation of systems whose complexity precludes a complete analytic study, and whose utilization of new components may make hardware breadboarding undesirable because of cost and procurement time limitations.

The present note describes software additions and modifications generated during the period July 1973 through March 1974 and is meant to be used in conjunction with the Signal Processing Interpreter Preliminary Description and User's Guide [1]. It replaces NUC TN 1213, by the same name [3].

TABLE OF CONTENTS

A. New Commands and Newly Implemented Commands

As the Signal Processing Interpreter has been used and tested by potential and actual users, the need for extensions and expansions of the system have become evident. These needs were mostly met by the addition of a variety of new commands to the repertory of existing commands.

Below is an alphabetic listing of these new additions giving the names only. Their functions and use will be covered in detail in the body of this report, under Section B.

'ADD'
'AUTOSCALE'
'BARKER GENERATOR'
'CIRCULANT PROJECTION'
'CLEAR'
'CLEAR X'
'COMPLEMENTARY SEQUENCE'
'CORRECT'
'CORRECT MACRO'
'DECIMATE'
'DEFINE DPCM GAIN'
'DEFINE DPCM QUANTIZER'
'DEFINE DPCM RCVR'
'DEFINE MACRO'
'DFT'
'DPCM'
'DPCM RCVR'
'EDCT'
'EXECUTE MACRO'
'EXPAND'
'EXPONENTIAL'
'IMAG PART'
'LIST DPCM'
'LIST MACRO'
'MASS RECALL'
'MASS STORE'

```
'MODIFIED LOG'
'NOISE GENERATOR'
'ODCT'
'PLOT'
'POLAR TO RECTANGULAR'
'PRINT ALL COMMANDS'
'QUANTIZER'
'REAL PART'
'RECIPROCAL'
'RECTANGULAR TO POLAR'
'RIGHT CIRCULAR SHIFT'
'RIGHT SHIFT'
'ROLL DOWN'
'ROLL UP'
'SCALE'
'SUM'
```

B. Detailed Description of the New Commands and Newly Implemented Commands*

1. Exponentiation

Exponentiation of the elements of a vector was made available to users of the Signal Processing Interpreter by the new command

'EXPONENTIAL'

It simply replaces each component Zi of the vector by e^{Zi}, where Zi is complex.

2. Reciprocals

The command 'RECIPROCAL' was added. It checks for zero data points of a vector to avoid illegal division formulations. If one is encountered, an error message results that says:

"VECTOR CONTAINS A ZERO. RECIPROCAL NOT PERFORMED."

and, in fact, no reciprocal is then performed. If all data points are legitimately non-zero, then processor replaces each point of the vector by its complex reciprocal.

3. Addition of Vectors

Addition of vectors was made possible by including the command 'ADD'. This will cause addition of the lowest two vectors in the stack, after zero filling the shorter one to the size of the longer. It will also lower the remaining vectors of the stack one level. Resultant is placed in the X-register.

*In the following paragraphs that describe the commands, use is made of a set of sub paragraphs under the headings "Programming Considerations". These were designed for the person who must know in detail how and why the code was written the way it was. The average user can either skim them or skip them entirely.

4. Shifts

a. Commands

There are two basic kinds of shifts available to the user of the Signal Processing Interpreter. There is a straight shift of the elements of a vector, with the end values being lost as they are shifted off the end and zeros filling the new positions opened up. This is called by:

'RIGHT SHIFT'

And there is a circular shift of the vector components, with all values being preserved. This is called by:

'RIGHT CIRCULAR SHIFT'

There are no 'LEFT SHIFT' or 'LEFT CIRCULAR SHIFT' commands. These cases are handled in another way. (See below).

b. Use

(1) To perform a right shift, user puts his vector into the X-register and calls 'RIGHT SHIFT'. Processor then interrogates him with message "ENTER NUMBER OF POSITIONS TO BE SHIFTED", to which he responds by entering an integer. Vector is then shifted that number of places to the right, while that number of vector components are lost off the right end. Also the left end will now contain that number of zero components. Shifted vector will appear in the X-register.

(2) To perform a left shift, everything is done the same way except that the number entered by the user for positions to be shifted, is entered as a negative integer. In this case the zero components appear at the right end and shifted components are lost off the left end.

(3) To perform a right circular shift, user puts his vector in the X-register and calls 'RIGHT CIRCULAR SHIFT'. He receives the same message

"ENTER NUMBER OF POSITIONS TO BE SHIFTED"

to which he responds by entering an integer. A circular shift then takes place and resultant appears in the X-register.

(4) To perform a left circular shift, everything is done the same way except that the number entered by the user for positions to be shifted is entered as a negative integer.

(5) If the number entered for either shift command is zero, nothing takes place and user is asked for next command.

(6) If the number of places shifted is greater than or equal to the vector size for a straight shift, the result is to zero out your vector.

Programming Considerations

(1) Circular shifts are done modulo the vector size. A circular shift by the vector size leaves the vector unchanged.

(2) Circular shifts are done by a FORTRAN subroutine called CSHIFT, while straight shifts are done by Code embedded in ALGOL routine SPIN 3.

(3) In order not to cause problems, the value used for a straight shift is made equal to the smaller of the following two values: vector size and number entered.

(4) Since a circular shift by a multiple of NX is equivalent to a shift of zero, a circular shift of -k is equivalent to a circular shift of NX-k, so the distinction between right circular shifts and left circular shifts is purely conceptual.

5. Stack Operations

a. Commands

Several new stack operations were provided and are listed below:

'ROLL UP'
'ROLL DOWN'
'CLEAR'
'CLEAR X'

b. Use

(1) User would use these in same way as the stack operations on the HP-35. Their usage is straightforward. See reference [7] for information on the HP-35.

(2) 'CLEAR' initializes with zero all 512 cells of all registers, regardless of their original vector dimensions. Then it sets all register dimensions to the nominal value of 1. 'CLEAR X' does so only for the X-register.

(3) 'ROLL UP' and 'ROLL DOWN' also work on 512 cell transfers regardless of vector size.

c. Programming Considerations

(1) The vectors were cleared as 512 and rolled up or down as 512 for two reasons. One was efficiency of code written and the other to avoid leaving garbage in cells that might give trouble at a later time.

(2) 'ROLL UP' and 'ROLL DOWN' actually perform cyclic permutations of the vectors X, Y, Z, T and also of their dimensions NX, NY, NZ and NT.

6. Scaling

a. Commands

Scaling can be done to a vector by using one of the following commands:

'SCALE'
'AUTOSCALE'

The first one requires a scale factor to be supplied by the user, while the second one is an automatic scaling process.

b. Use

(1) The commard 'SCALE' calls upon the user to

'ENTER COMPLEX SCALE FACTOR'

When he does, his original vector is multiplied pointwise by this scale factor, and the result is in the X-register.

(2) When 'AUTOSCALE' is called, processor calculates the absolute value of the largest point of the original vector and scales vector by this quantity. It also prints out the message

"PREVIOUS LARGEST ABSOLUTE VALUE = _____" .

The variable printed is this quantity used for scaling.

c. Programming Considerations

(1) A FORTRAN subroutine called SCALE was used even though the code was so short to suppose that it would logically be embedded in the ALGOL SPIN 3 procedure. This was done to allow uniform entry of the complex scaling factor. To enter complex numbers by FORTRAN is different from ALGOL. An entry of a complex number for FORTRAN is of the form

4., -3.6

while for ALGOL, it is of the form

4. -3.6

(2) Autoscaling makes use of the ALGOL procedure MAX.

7. Correction of Vector Points

a. Command

The user may correct a point of a vector in the X-register by calling

'CORRECT'

b. Use

(1) After user places his call to 'CORRECT', processor comes back to ask user to

"ENTER INDEX OF POINT TO BE CORRECTED" .

(2) User then enters his integer and receives back the message

"ENTER COMPLEX DATA POINT NUMBER _____" ,

where the variable is the index he just entered.

(3) Now he enters a complex value in FORTRAN format and receives a verification print back of what he just entered, with the message

"VALUE STORED AT INDEX ____ IS _____" .

(4) Processor replaces previous value at that point by the new one.

c. Programming Considerations

(1) Correction is done in a FORTRAN subroutine named CORRV. This was done to allow use of standard FORTRAN format for inputing the complex value of the point to be corrected. ALGOL uses a different format, and it was necessary to have consistency of input conventions.

(2) At some later time, it may be desirable to make sure index entered is within the bounds of the vector being corrected. No check is now made and problems could result from sloppy use.

(3) Complex input format for FORTRAN has the real part first, then a comma, then the imaginary part. The real part and the imaginary parts are real numbers with decimal points.

8. Printing Command Names

a. Command

An operator was implemented to allow the printing out of the command name as soon as it is recognized. It is called by

'PRINT ALL COMMANDS' .

b. Use

(1) Whenever, if ever, the user desires to see the command names printed out, he need only call once, 'PRINT ALL COMMANDS', and it will be done for the remainder of the run.

(2) This was implemented mainly for batch jobs where a sequential listing of the commands performed is otherwise unavailable.

(3) This printing of commands will also occur for the elements that make up a macro.

c. Programming Considerations

(1) Processor uses a flag called PCFLAG. It has an initial value of 0 and changes to 1 at the first time 'PRINT ALL COMMANDS' is called. Thereafter it remains 1 for the rest of the run. In other words, there is no way to shut it off, at present. An additional command could easily be created to do that. It need only reset the flag.

(2) This command provides a printout for batch jobs which is readable without reference to the card deck which was used for submission of the job.

9. Polar and Rectangular Coordinate Conversion

a. Commands

User can perform conversion of a vector from polar to rectangular coordinates by calling:

'POLAR TO RECTANGULAR'

Or he can convert rectangular coordinates to polar by:

'RECTANGULAR TO POLAR'

b. Use

When either of the above is called, the user receives message:

"ENTER R OR D IN STRING QUOTES FOR ANGLES IN DEG, RAD"

User responds with 'D' or 'R' .

c. Programming Considerations

(1) Vector data is complex type, so when rectangular, the real part is X coordinate and imaginary part is Y coordinate. For polar, real part is magnitude and the imaginary part is angle.

(2) Program actually only checks for the 'D' option. If not 'D', then 'R' is assumed.

(3) It was necessary to include a special routine to compute the angles. The ALGOL arctangent routine does not preserve uniqueness through the transform since it uses only a single input argument. The FORTRAN routine FUNCTION CANG was added. It also handles the case of zero for both coordinates. To use it however, it was necessary to type it as

EXTERNAL FORTRAN REAL PROCEDURE .

10. DFT

a. Command

User may call 'DFT' when a discrete Fourier transform is desired and not have to make the decision himself about whether it is an 'FFT' or a 'CZT'.

b. Use

User puts vector to be transformed into the X-register, calls 'DFT', and receives his transformed vector back in the X-register.

c. Programming Considerations

(1) Program will check the vector dimension for powers of 2 up to an exponent of 9, that is, up to a value of 512. If a power of 2, then the FFTSP subroutine will be called. Otherwise it will be CZT.

(2) Code is redundant with some in 'EDCT'. (See writeup of Cosine Transforms). It may be desirable to convert this to a subroutine and replace the redundant code with subroutine calls.

11. Noise Generation

a. Command

A noise generation capability is available to the Signal Processing Interpreter user by calling

'NOISE GENERATOR'

b. Use

(1) After calling 'NOISE GENERATOR', user receives back the message:

"ENTER INITIALIZATION POINT FOR NOISE GENERATOR" .

(2) He responds by entering a positive integer that controls the starting point of the random number generator that creates the noise sequence.

(3) User then is asked to: "ENTER VARIANCE". This is a real number and represents the variance that the final noise sequence will have.

(4) After he enters this number, the noise sequence is generated and appears in the X-register.

(5) The length of the noise sequence generated is the length of the X-register vector at the time it is called.

c. Programming Considerations

(1) Processor calls upon a FORTRAN subroutine called NOISE to perform the task.

(2) The method used for the random number generator came from "Communications of the ACM", Volume 12, Number 2, of February 1969, p. 93-94, by J. B. Kruskal. [2]

(3) The random number generator used was set up to allow sequences of up to 8192 points. Since we are limited to 512, in general, we can start the sequences at a variety of points to get independent sequences. Method generates a zero-mean random vector using a multiplicative congruential method.

12. Vector Expansion and Decimation

a. Commands

Expansion and decimation of a vector are available to the user through the commands:

'EXPAND'
'DECIMATE'

b. Use

(1) To expand a vector in the X-register, user calls 'EXPAND'. Processor then asks user to

"ENTER INTEGER EXPANSION FACTOR" ,

to which the user responds with a positive integer. (Call it INT)

(2) The new vector will be of length

1 + ((NX-1) * INT), where NX was the original length. The first point remains fixed, while between each two points of the original vector there will be inserted (INT-1) zero points. Resultant appears in the X-register and destroys the original vector.

(3) Decimation works in a corresponding way. With the vector in the X-register, the user calls 'DECIMATE'. The processor then asks user to

"ENTER INTEGER DECIMATION FACTOR" ,

to which the user responds with a positive integer. (Call it NNT).

(4) The new vector will be of length 1 + ((NX-1)/NNT) where NX was the original vector size. The new vector will be the first point and every NNT-th point thereafter.

(5) Note that these operators are not commutative, either in value or in length.

c. Programming Considerations

(1) The code for decimation uses integer division as noted by the // sign. This is necessary to truncate off fractional parts.

13. Cosine Transforms

a. Commands

Even and odd discrete cosine transforms are available through the commands:

'EDCT'
'ODCT'

b. Use

(1) User puts vector to be transformed in the X-register and then gives appropriate call: 'EDCT' for even discrete cosine transform or 'ODCT' for odd discrete cosine transform.

(2) Processor generates transform and stores the results back into the X-register.

(3) Size of the original vector is limited to 256, with the one exception of the number 512 being allowed for the 'EDCT'.

c. Programming Considerations

(1) Actual formulas implemented are not included here nor is an explanation of the method. User is referred to NUC TN 1265, entitled "High Speed Serial Access Implementation for Discrete Cosine Transforms" by J. M. Speiser.

(2) The reason for limitation of size of original vector is due to method used. If not a power of 2, method makes use of CZT routine which quadruples size. But the dimensioning is limited to 1024, thus 256 for original vector.

(3) Routines make use of a flag called NODD, which takes on the value 1 for odd or 0 for even. The odd transform automatically uses the CZT routine, while the even one must test for power of 2. It it is a power of 2, then the FFT routine is used; otherwise the CZT is again used.

(4) We cannot use the 'DFT' formal call to do this as the code is embedded in the SPIN 3 routine, but the code to do this checking is identical. One, at some later time, could make the 'DFT' a subroutine and then use it for both places, as the code is redundant.

14. Modified Log

a. Command

A modified logarithm routine is available to user with call

'MODIFIED LOG'

b. Use

(1) Vector to be processed is placed in the X-register and call is made to 'MODIFIED LOG'

(2) At this point program interrogates user with message

"ENTER MINIMUM RECOGNIZED NUMBER'

(3) User then enters a positive, real number. If it is not positive, a diagnostic results

"ERROR. NUMBER MUST BE POSITIVE REAL"

and he is again asked to enter his number until he gets it right.

(4) The real part of the vector is then checked against this minimum recognized number. If it is larger, then the complex number is replaced by the natural logarithm of its real part. If not, it is replaced by the natural logarithm of the minimum recognized number.

c. Programming Considerations

(1) Code will not at present catch errors in type for entering the minimum recognized number. But this safeguard could easily be built in.

15. MACROS

a. Commands

User may define and use combinations of Signal Processing Interpreter statements as blocks or macros. To do so he will use some or all of the following four commands:

'DEFINE MACRO'
'LIST MACRO'
'CORRECT MACRO'
'EXECUTE MACRO'

b. Use

(1) To begin the process, user enters 'DEFINE MACRO'. Only one macro may be used at one time by the user. The processor then prints out:

"ENTER NUMBER OF COMMANDS TO BE USED FOR MACRO."

The user then enters an integer between 2 and 20. (Actually 1 would work, but would be meaningless). If he enters a number greater than 20, a diagnostic results.

"ERROR. MORE THAN 20 COMMANDS"

He is again asked for number of commands.

(2) Once he has properly entered the number, he receives the message:

"ENTER COMMAND" until he has entered enough statements to make up the desired size for the macro. At this point he receives message:

"END OF MACRO DEFINITION"

(3) Provisions are made to keep the run from terminating if user makes a bad entry. He then receives a message:

"ERROR IN ENTERING LAST MACRO COMMAND"
and is given another chance to enter it correctly.

Note: Processor does not do a command table search at this time so command still may not be legitimate. It just means that entry was of correct format.

(4) User may check to see what is in the macro by calling:

'LIST MACRO'

It gives a printed listing of the command names that make up the macro, and tnis process makes a good check.

(5) The user may desire to correct the generated macro or change it. He may do so by calling

'CORRECT MACRO'

This is similar in concept to the command 'CORRECT' associated with changes in vector values.
Program returns message:

"ENTER INDEX OF COMMAND TO BE CORRECTED"

User enters an integer between 1 and 20 and receives back message: "ENTER COMMAND" , which he does. Process repeats if an error occurs in entering. Upon successful completion, control returns to normal sequence.

(6) User is now ready to execute his macro, done by calling:

'EXECUTE MACRO'

This will cause the first command to be executed and all subsequent others possible up to the first required parameter input or option selection. Process continues until all commands in macro have been executed, at which time user receives message:

"END OF MACRO EXECUTION"

(7) Control returns to normal sequence.

(8) User note that the normal sequence message of the form

"NX=8 PLEASE ENTER NEXT COMMAND"

will be deleted between executed commands of a macro. It <u>will</u> appear after the completion of any one of the four macro control commands.

(9) Macro, once generated, is available to the user, at his discretion, for the rest of the run.

c. Programming Considerations

(1) Macros are limited to 20 commands at present. This could easily be changed by minimal reprogramming. Also more than one macro could be implemented by a somewhat larger job of programming.

(2) Read error checks are built in to catch errors in entering commands to macro. Check is made for form only. Errors in spelling or erroneous names won't be caught until user tries to execute his macro.

(3) A flag named MAFLAG is used to control suppression of unwanted print statements between macro commands. It is set to 1 during execution of a macro and to 0 at all other times.

16. Mass Storage

a. Commands

Mass memory for vector storage and retrieval is available to the signal processing interpreter user through the following commands:

'MASS STORE'
'MASS RECALL'

b. Use

(1) In order to use this storage capability, the user must include in his preliminary setup commands (before execution) the following two statements (or variations of them):

@ASG, AZ NAME.
@USE 15, NAME.

where NAME is the users own catalogued file, previously assigned by:

@ASG, UP NAME., F/120/TRK

The 120 tracks of mass storage file are necessary to allow for a maximum of 201 vectors of 512 complex words. User should supply his own file name for NAME.

(2) Now the Signal Processing Interpreter can be executed.

(3) The two storage commands are very similar in use. To store a vector, the user puts his vector into the X-register and types 'MASS STORE'. The Signal Processing Interpreter then interrogates him with the message:

"ENTER INDEX OF MASS-STORED VECTOR" .

User then enters an integer between 0 and 200. The program will store the vector in the appropriate memory area along with its size. Conversely, to retrieve a vector, user types

'MASS RECALL'

at which time he receives the same interrogation as above. When he enters the integer ($0 \leq N \leq 200$), he receives the vector back in the X-register along with its size in NX. User can store up to 201 vectors at a time. 'MASS RECALL' treats the stack in the same way as 'RECALL', that is, it raises the stack one level with the topmost vector being lost.

c. Programming Considerations

(1) Program uses same code to accomplish both commands, with a read-write flag as the only difference.

RWFLAG = 1 for write and RWFLAG=2 for read.

(2) Program makes access to mass storage by sector addressing. In order to do so, program computes and locates proper spot by using internal calculations in SUBROUTINE MASTOR and by calling system SUBROUTINE SETADR with correct index.

(3) To make the above calculation one needs to know that there are 28 words per sector. In order to be able to store a 512 complex word vector, we would need at least 1024 words. This will be handled by 37 sectors (1036 words). In addition, the program writes and stores a one-word block, before storing the vector, that gives vector length. This requires an additional sector for a total of 38.*

(4) If an index of 200 (actually the 201st position) is exceeded, user is given a diagnostic message:

"EXCEEDED MAX INDEX. OPERATION NOT DONE."

Then the mass storage operation is not performed, with control returning to ask the user for the next command.

(5) The NTRAN no-op statement before CALL SETADR of SUBROUTINE MASTOR is required in order to assign unit 15 as an NTRAN unit. If it is not present, the call to SETADR assumes 15 is a FORTRAN file, and then the first NTRAN call to unit 15 causes the run to bomb. (A file cannot be both FORTRAN and NTRAN).

*There are 64 sectors to 1 track. Since we need 201 vectors each of 38 sectors length, we need a total of 7638 sectors, or about 120 tracks.

17. Circulant Projection

a. Command

User may generate a circulant projection of a given vector by calling:

'CIRCULANT PROJECTION'

b. Use

(1) This replaces X_p by $(1 - \frac{P}{N})X_p + \frac{P}{N} X^*_{N-P}$

for P = 0, 1, 2, ..., N-1 and where * means complex conjugate.

(2) User puts his vector to be processed in the X-register, calls 'CIRCULANT PROJECTION', and then receives this projection back in the X-register.

c. Programming Considerations

(1) Code makes use of several procedures already defined, as well as several temporary arrays for intermediate results.

(2) Formula above supplied by Jeff Speiser.

18. Barker Sequence Generation

a. Command

User may generate a Barker sequence by a call

'BARKER SEQUENCE'

b. Use

(1) By such a call, a Barker sequence will be generated in vector form in the X-register.

(2) Processor asks user to

"SELECT SIZE FROM 2, 3, 4, 5, 7, 11, 13" .

(3) User then enters size as an integer, one of the above. If he makes a mistake and enters some other integer, he will get either the 7 sequence or the 13 sequence, depending upon the size of the number he entered. But he is not kicked off.

c. Programming Considerations

(1) Barker sequences are from a book by C.E. Cook and M. Bernfeld. See reference [5]. The above numbers represent all known sequences.

(2) Where more than one sequence is known (i.e. for 2 and 4), only one of them is used.

(3) Code for generating sequences is contained in a FORTRAN subroutine called BARKER.

(4) In order to conserve core storage, sequences are overlapped where possible. The variable J gives the starting point index of the sequence in the input data array.

(5) Sequences are given using +1 and -1 rather than just + and - as in paper.

19. Complementary Sequence Generation

a. Command

A complementary sequence generation capability is available to the user by a call to

'COMPLEMENTARY SEQUENCE'

b. Use

(1) User can generate a complementary sequence of length 10, 26, or a power of 2 up to 9, that is, the number up 512.

(2) After calling 'COMPLEMENTARY SEQUENCE', the user receives back the message:

"SELECT SIZE 10, 26, or 0-9 (FOR POWER OF 2)"

(3) If he selects 10 or 26, a pre-stored sequence will be fed back into the X-register with the appropriate dimension. If it is a power of 2, then the sequence is generated by block recursion: $A' = AB$, $B = A\overline{B}$. See reference [6].

(4) Sequences form vectors of complex components.

c. Programming Considerations

(1) Actual sequence generation takes place in a FORTRAN subroutine called COMPLM.

(2) If user makes a mistake and chooses some other integer than those allowed, he will get either the 26 sequence for numbers greater than 10 or he will get the sequence for 2 to the power 1 for negative values. This is due to the way the tests are made. These will not be what he asked for, but he will not have his run terminated either.

(3) Real and imaginary parts of the 10 and 26 sequences are stored in real arrays and assembled by using the FORTRAN function CMPLX.

(4) Sequence elements are made up of +1 and -1 rather than 1 and 0 as given in reference [6].

20. Differential Pulse Code Modulation

a. Commands

The Signal Processing Interpreter has four commands which make possible the definition and use of a differential pulse code modulation and two more commands for an associated receiver. These are:

'DEFINE DPCM GAIN'
'DEFINE DPCM QUANTIZER'
'LIST DPCM'
'DPCM'
'DEFINE DPCM RCVR'
'DPCM RCVR'

b. Use

(1) In order to use the 'DPCM' command or the 'DPCM RCVR' command, the user must first define them. He does so by using the other commands. If no definition takes place, then the gain is assumed to be of magnitude 1 and the quantizer, if present, is assumed to be of an infinite no. of steps. (That is, output = input.)

(2) User calls 'DEFINE DPCM GAIN' in order to define or redefine gain. Program then interrogates user with message "ENTER LOOP GAIN. REAL NUMBER BETWEEN 0. AND 1." , to which the user responds. Note that the default value is 1.

(3) The DPCM is still considered undefined until user makes first call to 'DEFINE DPCM QUANTIZER' . Then the following sequence takes place:

(a) Program interrogates user with message

"DEFINE QUANTIZER 1. ENTER NUMBER OF JUMP POINTS"

(b) User responds with an integer between 1 and 20.

(c) Program then asks for jump points to be entered with message:

"ENTER REAL NUMBER FOR JUMP _____" .

(d) User enters jump points in response.

(e) Same sequence as (a), (b), (c), and (d) for Quantizer 2.

(f) DPCM is now defined.

(4) If user wants to see how DPCM is defined, he can use

'LIST DPCM' .

This will print out gains and jump points for the two quantizers.

(5) To use 'DPCM RCVR', user must first call 'DEFINE DPCM RCVR' or he will get the default case of gain = 1.
When called, it interrogates user as in (2) above. When answered, DPCM RCVR is considered defined.

(6) Once defined, user need only call 'DPCM' or 'DPCM RCVR' to perform desired operation.

(7) Redefinition may take place at any time by the user.

c. Programming Considerations

(1) Internal flags are set the first time either DPCM or DPCM RCVR are defined. After that, they are assumed to be defined for the rest of the run, though redefinition may take place.

(2) Quantizers are limited at present to at most 20 points. But more could be easily added with simple changes to the program.

(3) When user tries to use DPCM or DPCM RCVR but has not defined them, he gets a diagnostic message along with having program use default conditions.

(4) If a run bombs off before completion and execution must be restarted, then redefinition must take place again. No provision is made for storing DPCM values or those for DPCM RCVR from run to run. Each run reinitializes all quantities involved.

21. Quantization

a. Command

User has available a quantizer by simply calling

'QUANTIZER' .

b. Use

(1) User puts his original vector in the X-register and calls upon 'QUANTIZER'. Processor immediately asks him to:

"ENTER INTEGER FOR QUANTIZER"

(2) As soon as user does so, his quantized vector is generated and entered back into the X-register.

(3) This command provides a uniform quantization into the specified number of magnitude levels with preservation of sign. Quantization step size is equal to the maximum of the real and the imaginary parts in the array divided by the number of levels.

c. Programming Considerations

(1) Processor makes use of a FORTRAN subroutine called QUANT to do the quantization.

(2) The scaling factor used is calculated by checking all the components of the vector for the largest real or imaginary component magnitude. Then the integer entered is divided by this maximum magnitude to make up the scaling factor.

(3) Much of the code in QUANT involves getting the rounding process correct. It must take place separately for real and imaginary parts and also handle negative values. The real and imaginary parts are rounded to the nearest integer.

22. Real and Imaginary Parts of Vectors

Commands to extract the real part or the imaginary part of a complex vector have been introduced with the calls:

'REAL PART'
'IMAG PART'

Note, however, that 'IMAG PART' replaces the X vector by its imaginary part but is stored then as a real vector. That is, the original real components are destroyed and the original imaginary components take their place by moving.

23. Summation of Vector Components

Summation of the points of a vector is available through use of the command 'SUM'. The resultant sum is placed in the only point of a one-point vector in the X-register, which replaces the vector being summed. Original vector is lost. This sum is a complex number.

C. Modifications to Existing Commands

1. Setting Dimension

a. Command

User may set vector dimension by using:

'SET X DIMENSION'

b. Use

(1) After giving command call, user receives back the message:

"ENTER DESIRED DIMENSION"

To answer this, he enters an integer between 1 and 512.

(2) If a mistake is made entering the integer, such as an alphabetic character, then user is given diagnostic message:

"ERROR MADE IN ENTERING DIMENSION. TRY AGAIN."

And sequence starts over again with message to:

"ENTER DESIRED DIMENSION"

(3) When dimension is properly entered, control returns to ask for next command.

c. Programming Considerations

(1) Code makes use of error conditions for a READ statement. These will handle almost all errors except an entry with an @ sign, signifying an 1108 control statement. This causes a drop in level and termination of execution.

(2) In order to use the error condition, it is necessary to define an error exit label. One cannot just define it with a colon but must also include the label name among the non-executable statements, under LOCAL LABEL.

2. Inputting Vectors By Points

a. Several changes were made to facilitate the command 'INPUT VECTOR'. The interrogation of the user was transferred entirely to the FORTRAN subroutine called READV.

b. Next READ error checks were built into the reading-in of the complex data points. Errors caused by bad inputting of data will no longer terminate a computer run but will simply cause a diagnostic message to be printed out.

"ERROR IN ENTERING DATA POINT" .

Then the user will be given another chance to enter his point correctly. Most errors will be caught but not ones in which the user begins with a master space, i.e. @. These will cause a drop in level, and user must begin his run all over again.

c. The effect of this is that noise on the telephone link or user format errors will now only result in an error message and a request to re-enter the command or data point.

d. Programming note: it became necessary to read the subscript of the input variable with the variable value II+1 rather than the normal loop variable value I which you would expect. This was due to a FORTRAN compiler error discovered while trying to use I. Details as to what the error did will not be covered here. It suffices to say that the error was verified by the System Programming Group and forwarded on to UNIVAC for correction via an SSFR by Ben Ermance.

3. Plotting Vectors

There are now two recognized calls to obtain plots of the X vector. Both call upon the same code, and the second call was created only for the convenience of the user. They are:

'PLOT COMPLEX VECTOR'
'PLOT'

See reference [1] for more information.

4. Constant Functions

The routine 'CONSTANT FUNCTION' was rewritten to allow a consistent convention for input of complex values. Delimiter is now a comma, not a blank space as before. If more details are needed, see paragraph B.6.c.(1). The constant function merely fills the entire vector with a user-supplied complex constant.

5. Printing Vectors

In the past a number of routines automatically produced printed listings of the present vector contents. This has been changed to a philosophy of only doing what the user asks. Now only two routines will result in a printed listing:

'PRINT VECTOR'
'PLOT' or 'PLOT COMPLEX VECTOR'

In addition, when using 'PRINT VECTOR', you no longer get back the message saying "FINISHED PRINTING VECTOR." It was superfluous.

6. Other Minor Changes

a. Normalization of the FFT was by the square root of N. The CZT uses the FFTSP subroutine, and thus its normalization is automatic. Normalization is not now done within the CZT routine itself. Due to possible scaling problems, this procedure will probably be changed in the near future.

b. The following routines were rewritten for more efficient coding but have no substantial changes to their basic logic:

SUBROUTINE CHIRP
SUBROUTINE CPLOT
PROCEDURE CZT
SUBROUTINE REV
SUBROUTINE SEE
SUBROUTINE TONE

One change of note was the addition of EQUIVALENCE statements to SUBROUTINE CPLOT to eliminate the need for the FORTRAN functions REAL and AIMAG. This is far more efficient.

D. Changes to SPIN3 Processor System

A number of changes were made to the SPIN3 procedure, and the major ones will be mentioned below.

a. Previous input/output inconsistences due to the differences between FORTRAN and ALGOL conventions have been eliminated. This mainly affected the inputting of complex data points.

b. All vector indices, as far as the user is concerned, are now consistent. They run from zero to N-1 and appear in inputting, listing and plotting of vectors.

c. Errors that previously occurred when entering commands will no longer bomb the user off the machine. READ error checks were built into the system to handle these cases. This kind of error now causes the printing out of an error diagnostic message

'COMMAND NOT RECOGNIZED"

and then a return to ask the user to enter his next command. One error that still is not caught and corrected is when user includes the master space, i.e. @, as part of the command. This will cause a drop in level and termination of the run being executed.

d. The output printing has been cleaned up to minimize unnecessary line feeds, inefficient print statements and the printing of redundant information. Controls have been put into the processor, via the MARGIN procedure of the ALGOL processor, to delete all blank lines at the top and bottom of the print pages. This was done to allow plots to be continuous, as well as listings. This change affects batch jobs only. Clearly cathode ray scope terminals are not affected, but neither are the DCT 500 terminals, unfortunately. This is due to line-feeds hard-wired into the DCT 500 hardware. The MARGIN call is

MARGIN ('M,66,0,0 . ')

The M option is to readjust page length and top and bottom margins. The 66 is for a standard page, and the two zeros represents lines to be skipped at top and bottom of page. The string quotes are necessary since, in reality, the operator MARGIN has only one argument, namely a control string. And string must be terminated by a period. This takes place

at the time all initialization for the processor is done. See reference [4] for more details on use of MARGIN.

e. Much of the internal code has been rewritten for efficiency of execution and compactness of format. Thus even though the capabilities of the System have more than tripled, the number of punch cards has remained roughly the same. All the DO loops of SPIN3 were redone in compact form as were many IF statements, formats, etc. Procedures internal to SPIN3, called DOW, PAD, UP, and COPY were greatly compacted.

f. An initialization process was introduced into the processor to allow a variety of operations to be performed only once during the run to set up the conditions and prepare for execution of the remaining parts. Prior to this, every time a new command was input, the processor went back and reloaded all its command tables, in an ALGOL procedure called ATAB. It then went through an elaborate process involving Boolean logical operations to perform a test for matching. ATAB has been eliminated now, the command tables are loaded only once, the Boolean operations have been replaced by a simple IF statement, and all initialization is now done only once. This was done by putting much of the ATAB code into the SPIN3 processor itself, by making all initialization the first executable code in the routine and by always returning to points below this code. These changes improved running times for the executable portion of the program by a factor of between 4 and 5 times. Another change contributing to this great time saving was terminating the table search as soon as a match was made. Previously the entire table was searched, even if recognition had already taken place.

g. The basic flow of the processing of a command is now slightly more complex since the case of macro commands must be considered. Commands that make up a macro take a different path upon being recognized.

h. Extensive use was made of the formatted ALGOL WRITE statement. This allows literals and variables to appear upon the same line of print. It also allows multiple use of a format by different WRITE statements. These are non-executable statements that are placed with the other such declaritive statements at the front end of the procedure. A programming note: it is

necessary to activate each format statement at its end with an activation code that controls line feeds. Failure to include the activation code will have the effect of deleting the WRITE statement execution. In this sense, the format statements are entirely different from their counterparts in FORTRAN. See reference [4] for more details.

i. In addition, a great many changes were made to the sections of the processor, containing the code for the individual commands but these will not be covered here since this was already done in Section B.

E. References

1. Speiser, J. M., Signal Processing Interpreter Preliminary Description and User's Guide, NUC TN 1065, 25 June 1973.

2. Kruskal, J. B., Extremely Portable Random Number Generator, Communications of the ACM, Volume 12, Number 2, Feb. 1969, pp. 93-94.

3. Wood, B. R. and Speiser, J. M., Signal Processing Interpreter: Additions and Modifications, NUC TN 1213, 17 October 1973.

4. UNIVAC 1108 Multi Processor System - ALGOL Programmers Reference, UP-7544, 1967.

5. Cook, C.E. and Bernfeld, M., Radar Signals, An Introduction to Theory and Applications, Academic Press, 1967, p. 245.

6. Golay, M.J.E., Complementary Series, I.R.E. Transactions on Information Theory, April 1961, pp. 82-87.

7. HP-35 Operating Manual, 00035-90008, Hewlett Packard.