NASA CR-166726

(NASA-CR-166726) SAR PROCESSING ON THE MPP N82-11801
Final Report, 9 Jan. - 31 Aug. 1981
(Goodyear Aerospace Corp.) 88 p
HC A05/MF A01 CSCL 09B Unclas
G3/61 01619

SAR PROCESSING ON THE MPP

GOODYEAR AEROSPACE CORPORATION
1210 Massillon Road
Akron, Ohio 44315

August 1981
Final Report

Prepared for

Goddard Space Flight Center
Greenbelt, Maryland 20771

SAR PROCESSING ON THE MPP.

K, E. Batcher, et al

Goodyear Aerospace Corporation
Akron, Ohio

August 1981

N82-11801

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle	5. Report Date
SAR Processing on the MPP	31 August 1981
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
K.E.Batcher, E.E.Eddey, R.O.Faiss, P.A. Gilmore	GER-17020

9. Performing Organization Name and Address	10. Work Unit No.
Goodyear Aerospace Corporation 1210 Massillon Road Akron, Ohio 44315	11. Contract or Grant No. NAS5-26430
	13. Type of Report and Period Covered Final 9 Jan 81-31 Aug 81
12. Sponsoring Agency Name and Address H.K.Ramapriyan NASA Goddard Space Flight Center Greenbelt, Maryland 20771	14. Sponsoring Agency Code

15. Supplementary Notes

None

16. Abstract

The massively parallel processor (MPP) is designed to support ultra high-speed ground based image processing. The architecture comprises an array unit (ARU) which processes arrays of data; an array control unit (ACU) which controls the operation of the ARU and performs scalar arithmetic; a program and data management unit (PDMU) which controls the flow of data, and a unique staging memory (SM) which buffers and permutes data. The ARU contains a 128 by 128 array of bit-serial processing elements (PE's). Two-by-four subarrays of PE's are packaged in a custom VLSI HCMOS chip. The staging memory is a large multidimensional-access memory which buffers and permutes data flowing within the system.

Efficient SAR processing is achieved via ARU communication paths and SM data manipulation. Real time processing capability can be realized via a multiple ARU, multiple SM configuration.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Parallel Processors, Array Processors, Image Processing, Radar Data Processing, Synthetic Aperture Radar	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
None	None	46	

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1 - INTRODUCTION

1.1 General

This final technical report documents efforts and achievements by Goodyear Aerospace Corporation (GAC) under NASA Contract NAS5-26430, "Use of the Massively Parallel Processor (MPP) for an Advanced Digital Synthetic Aperture Radar Processor". The purpose of the contract was 1) to determine how well the MPP presently being assembled for NASA Goddard can execute SAR processing tasks; and 2) to identify what changes (if any) would increase its SAR processing capability. Under the contract GAC developed a conceptual system design for utilizing the MPP as a SAR processor and estimated system performance for a worst case mission. Performance evaluation was based on a SAR processing algorithm due to C. Wu.

The architecture of the MPP was found to be well adapted to SAR processing. An increase in array memory capacity (more bits per PE) will increase its SAR processing capability considerably. The staging memory in the MPP is very useful for corner turning and other data re-formatting operations.

1.2 Problem Overview

The processing of synthetic aperture radar (SAR) signals is of appreciable interest to NASA since many NASA missions will employ SAR sensors. An indication of the range of these missions is provided in Table 1 of Exhibit A to the contract statement of work. The mission set described by Table 1 is analyzed in detail with respect to processing load in Appendix A of this report. There it is shown that the ERSAR L mission out of the various missions provides the most severe processing load closely followed by the SEASAT mission. The baseline mission described in Exhibit A is quite similar to ERSAR L although it possesses the wider swath width of SEASAT. Consequently a processing system which meets the baseline mission requirements should readily have the capability to meet the other mission requirements.

The processing of SAR signals to produce an image represents a severe computing load since each image pixel results from lengthy convolutions in both the range and azimuth directions. Early digital SAR processors formed images by direct convolution methods. Since the development of the Fast Fourier Transform (FFT), attempts have been made to compute the SAR convolutions by means of FFT.

Convolution by means of FFT does reduce appreciably the number of computations required per image pixel. However other complications arise. The FFT process is a batch process; it is most efficient when many pixels are generated together. The batching operation in

turn is complicated by the need to account for range curvature. In the case of the baseline mission the range curvature covers seven range bins.

There have been a number of attempts to account for range curvature while still using FFT convolution. A particularly effective approach was suggested by Chialin Wu* (see Figure 1). The method is an FFT convolution approach. After the forward azimuth FFT operation, Wu obtains the composite spectrum of an image line by assembling the appropriate segments from the spectra of several azimuth lines. Wu used non-interpolative nearest neighbor selection of the spectra segments. However, in order to reduce azimuth side-lobes, interpolation over four range bins has been used in the case of the MPP algorithms to develop the composite spectrum, as described later in Section 2.

The MPP algorithms then employ FFT convolution procedures and use in general the Wu approach to range curvature correction. However the Wu approach has been modified to the extent of using interpolative spectral composition. The algorithms are illustrated in Section 2 of this report in application to the baseline mission. Further it was shown earlier in this item that the baseline mission encompasses the most stringent processing parameters of the Exhibit A, Table 1 SAR mission set.

1.3 Processing Overview

1.3.1 Processing Algorithm

As indicated in item 1.2 the processing approach chosen for the MPP is built upon the FFT based method developed by Wu. The overall MPP processing flow is shown in Figure 2. The first processing step is the range FFT which converts the signals from one radar pulse to the frequency domain. The length of this FFT must exceed the swath width in output pixels plus the range pulse length in complex samples. The value thus obtained is then increased by filling in zeros to make the total a value which is convenient for FFT processing, as discussed in Item 2.3 below. For the baseline mission this length was taken as 5120 points (5x1024). The 5120 in turn results from being the most convenient number greater than the sum of the 4000 pixel swath width and the 475 complex sample range pulse length.

* Wu, Chialin, "A Digital Fast Correlation Approach to Produce SEASAT SAR Imagery", IEEE 1980 Intl. Radar Conf. Proc., Apr. 1980.

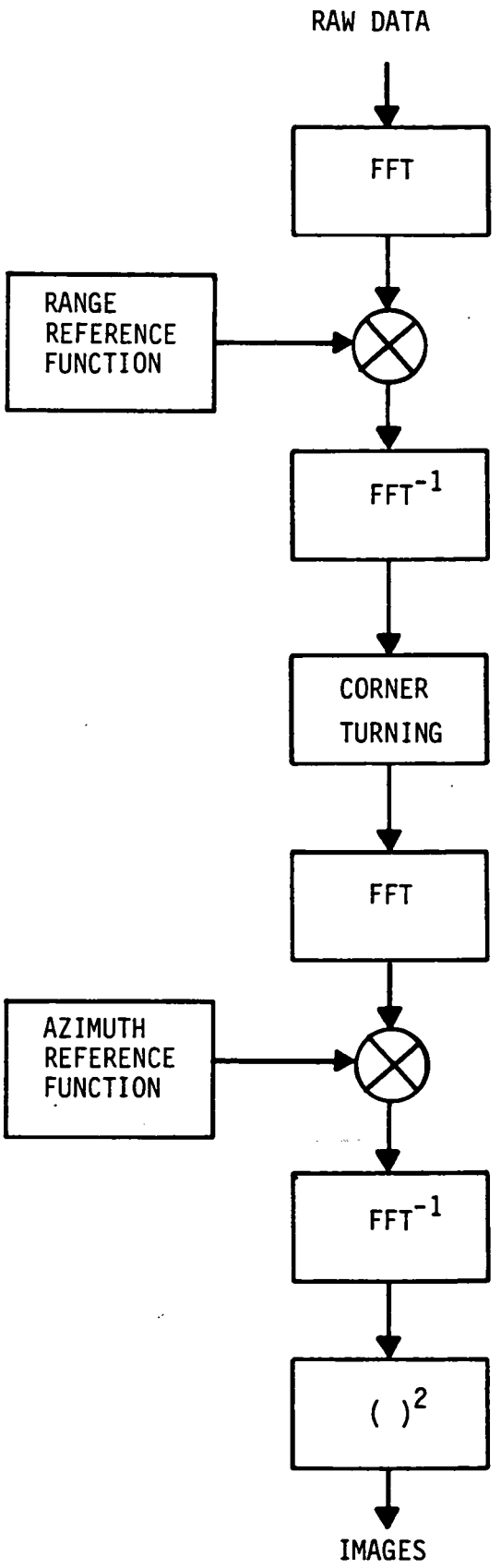

Figure 1 - FFT SAR Data Processing Approach Suggested by C. Wu

GOODYEAR AEROSPACE

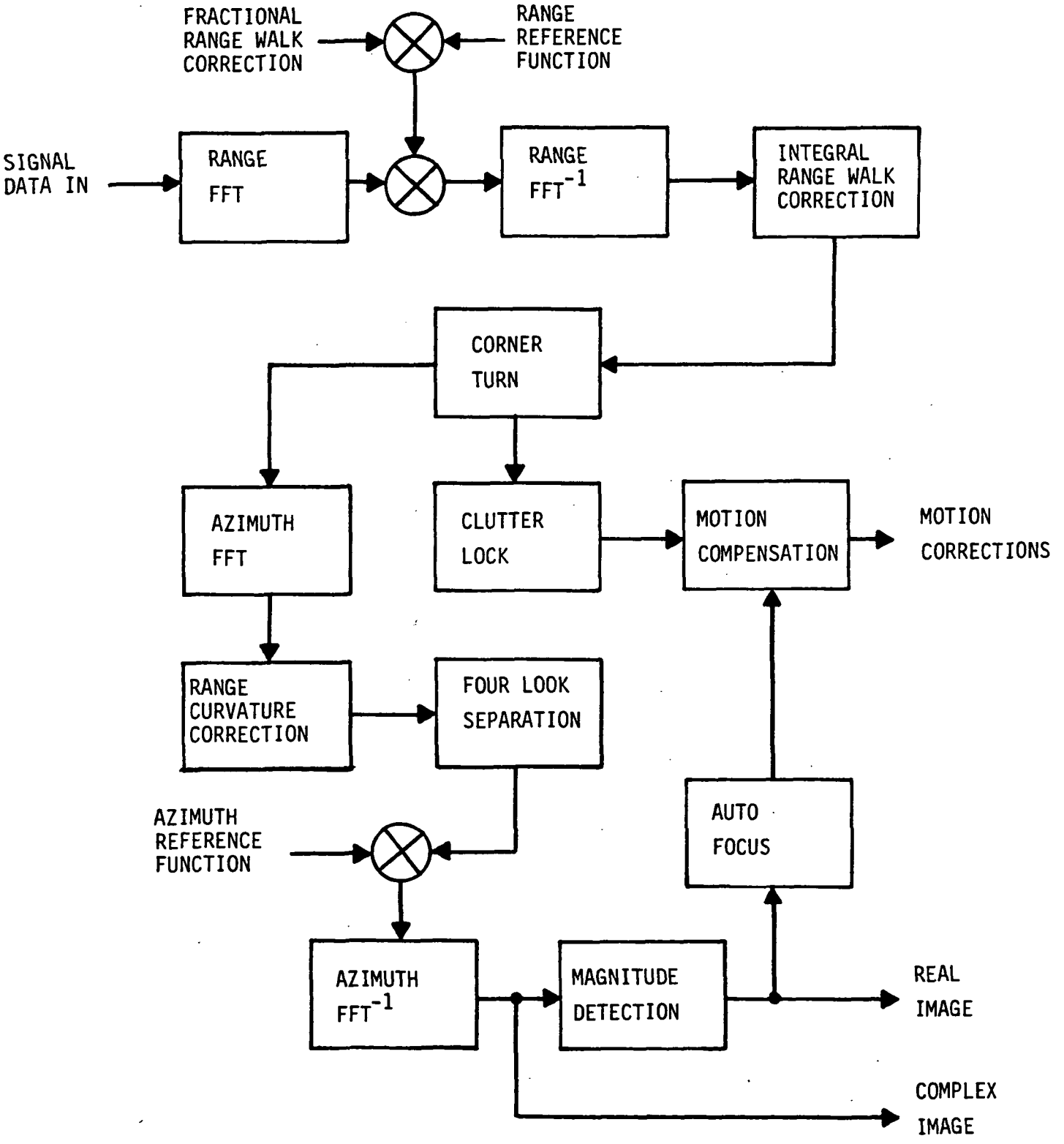

Figure 2 - Processing Algorithm Flowchart

GOODYEAR AEROSPACE

The output of the range FFT is then complex multiplied by a reference function. This function in turn is the product of the range compression function and a factor which accounts for the fractional portion of the range walk correction. This later factor is a linear phase shift term based upon the Fourier transform pair

$$f(t-T) \quad <==> \quad F(w) \exp(-iwT)$$

where T is the time delay needed to account for the fractional portion of the range walk correction. The integer portion of this correction is accounted for by an actual shift of the data in range from pulse to pulse.

The complex multiply operation is then followed by the range inverse FFT and the integer range walk correction shift. Next the corner turn needed to arrange the data for processing in the azimuth direction is performed. The azimuth FFT operation follows.

The length of the azimuth FFT must be greater than that required by the azimuth compression ratio times the number of multiple looks. Actually the length must be appreciably longer than this value to permit batching for efficient processing. The situation is further complicated however by memory requirements.

The processing required for an N point FFT is roughly proportional to N * log2 N. If the aperture length for the number of multiple looks is n, then the number of image points produced is N-n. The work W(N) per image point is (N * log2 N)/(N-n). For n=1750, the value for the baseline mission, W(N) is a minimum for N equal to about 19,000. Such a large value of N would result in excessive memory requirements so that a smaller value of N must be selected for practical implementation. A value of N = 3520 was chosen for the baseline mission as representing a reasonable compromise between processing and memory requirements.

The azimuth FFT is followed by the interpolation operation necessary to correct for range curvature. As indicated in Item 1.2 this interpolation is over four range bins. This operation is followed by the separation of the multiple looks based upon frequency values. The remaining processing steps shown in Figure 2 comprise the azimuth reference function complex multiplication, the inverse azimuth FFT which produces the compressed complex image and the magnitude determination operation which produces the real image. All the steps are discussed in

greater detail in Section 2.

Also shown in Figure 2 are the clutterlock and autofocus operations which are needed to obtain motion compensation data from the image data. These steps are initiated by running the input tape through a few aperture lengths, obtaining the initiating values, and then rewinding the tape to start the image generating processing using the motion data thus obtained. The motion compensation data is used to modify the range and azimuth reference functions appropriately. The clutterlock and autofocus operations are discussed in detail in Items 2.7 and 2.8 respectively.

1.3.2 Processing System

The advanced digital SAR processing system (ADSPS) considered in this study is built around the Massively Parallel Processor (MPP).

The MPP is designed to process arrays of image data at high speed. It has 16,384 processing elements (PE's) in a 128 x 128 array. Redundant PE columns improve availability. Processing is bit-plane oriented to accomodate pixels and results of any length. Each PE has a variable-length shift register, a serial adder, a logic section and a 1024-bit RAM. The design will accommodate changes in RAM capacity. Basic cycle time is 100 nanoseconds. Addition of array elements has a speed of 6553 MOPS (Million Operations per second) for 8-bit elements, 4428 MOPS for 12-bit elements and 430 MOPS for 32-bit floating point elements. Multiplication speed is 1861 MOPS, 910 MOPS and 216 MOPS, respectively. Input and output of data through wide 128-bit ports can occur at speeds up to 1.28 billion bits per second. The array of PE's is controlled by an array control unit containing a micro- programmable PE control unit, a fast scalar processor (main control unit) and an I/O-control unit. A unique internal staging memory buffers and reformats data flowing within the system.

A minicomputer is used as a program and data management unit. It executes the program development software (micro-code assembler, main assembler, linker), manages the system and connects it to a host computer.

A block diagram of the MPP-based ADSPS is shown in Figure 3. The array unit of the MPP is connected to two staging memories. Staging memory 1 receives the input data and

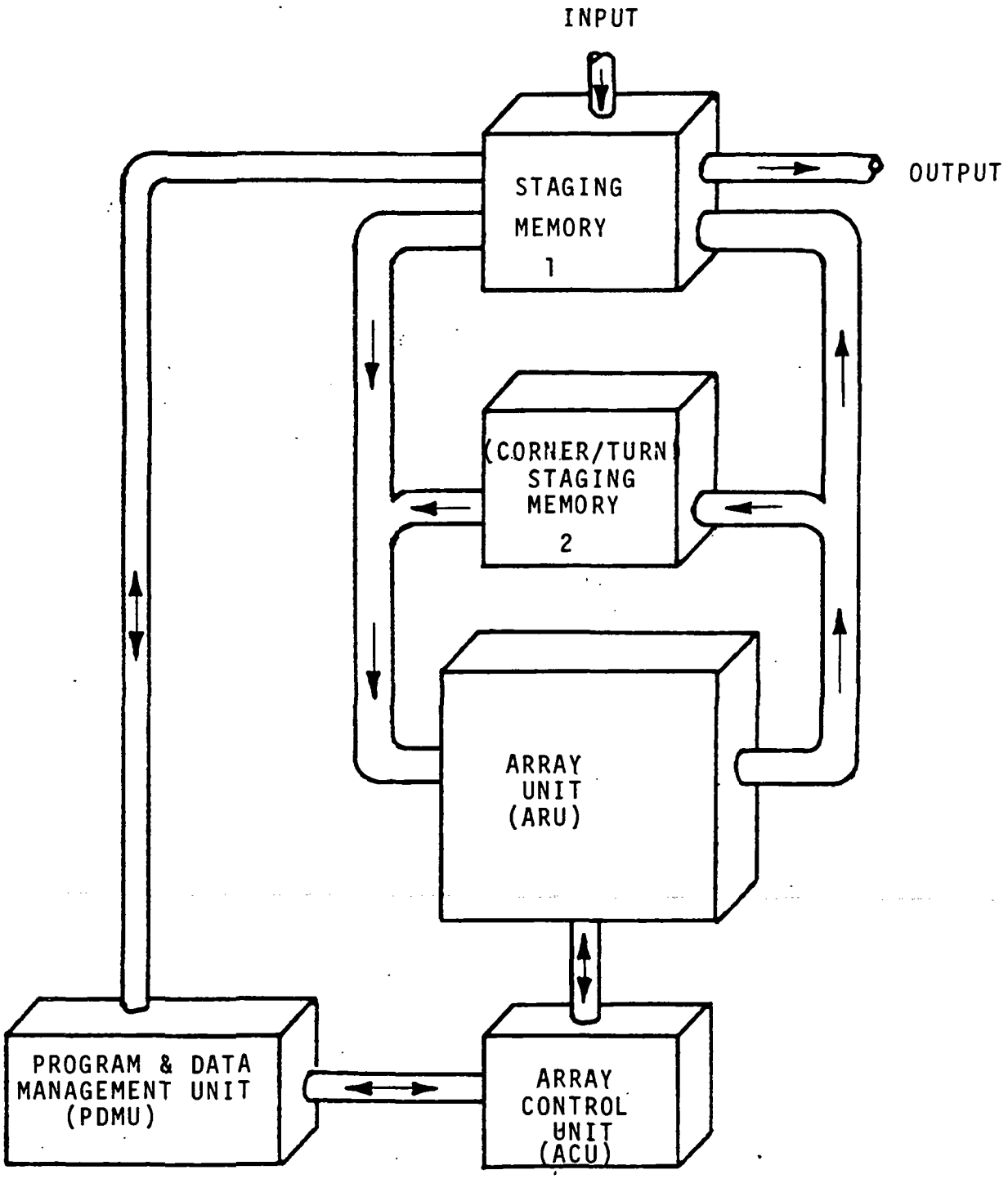

Figure 3 - Block Diagram of ADSPS

the final display data. Staging memory 2 is used for the corner-turning operation between the range processing and the azimuth processing. The array unit alternates between range processing and azimuth processing as described in item 1.3.3.

1.3.3 Processing Flow

The processing algorithm as flowcharted in Figure 2 is performed by alternating between the range and azimuth phases. The range phase includes the range FFT, the complex multiplication in the frequency domain, the range inverse FFT, and the integer range walk correction. The range-processed data samples are placed in staging memory 2 for the corner-turning operation. Initially, the range phase is executed for 3520 radar pulses to build up enough data in staging memory 2 for the azimuth phase. After executing the azimuth phase once, the array unit executes the range phase for 1760 pulses and the output data replaces the first half of the data in staging memory 2. Then the azimuth phase is executed again, followed by the range phase, etc.

The azimuth phase includes the azimuth FFT, range curvature correction, separation of the looks, complex multiplication by the azimuth reference function, inverse FFT, magnitude detection and combination of the looks for the display. The length of the azimuth FFT is 3520 + 576 zeroes = 4096. On each execution the input data is slipped by 1760 pulses so half of the input data is old data and half is new.

Range processing is described in item 2.4 and its sub-items. The array unit treats 32 radar pulses at one time. Range processing is executed 55 times to treat 1760 radar pulses. Initially, range processing is executed 110 times to treat the first 3520 pulses.

Azimuth processing is described in item 2.5 and its sub-items. The array unit inputs 32 range bins at one time and range curvature correction limits the number of output range bins to about 23. To handle the 4000 output bins requires 174 executions of the azimuth phase.

SECTION 2 - SAR PROCESSING ON THE MPP

2.1 Baseline Mission

As indicated in item 1.2 of Section 1 the baseline mission specified in Exhibit A of the contract statement of work is similar to the ERSAR L mission set although the swath width specified is greater than that of ERSAR L and more similar to that of the SEASAT mission set. In Appendix A of this report the following processing parameter values are developed for the ERSAR L mission set:

PRF per look	=	644 Hz
Radar PRF (4 looks)	=	2576 Hz
Time to travel Aperture Length	=	0.715 Second
Radar Pulses during Aperture Time	=	2576 x 0.715
	=	1841.84 pulses
Pulses for one look	=	460.46 pulses

Table 2 of Exhibit A gives the following:

Radar PRF	= 2500 Hz
Number of Azimuth Looks (concurrently processed)	= 4
Azimuth Bandwidth per Look	= 500 Hz
Azimuth Compression Ratio per Look	= 350

From these data it can be concluded that:

PRF per look = 2500/4 = 625 Hz

Time to travel Aperture Length = 350/500 = 0.7 Second

Pulses during Aperture Time = 2500x0.7 = 1750 pulses

Pulses for one look = 1750/4 = 437.5 pulses

The differences between the data developed from the ERSAR L mission and that provided for the baseline mission are small. The differences are probably caused by round-off operations or by the use of slightly different parameters (such as satellite velocity) for the calculation of various terms. The important fact is that the baseline mission parameters can be tied back to basic mission parameters very well. The baseline mission parameters from Table 2 of Exhibit A of the Statement of Work will be used in the analyses in the remainder of this report. Actually, small differences in the starting parameter values for the analysis will have little effect on the procedures or timing analysis since the FFT lengths have been increased from the exact required values to power of 2 related values which are more convenient for computation.

For the baseline mission the range FFT is a 5120 point transform

(5x1024) as previously indicated in item 1.3.1. The minimum value actually required is 4000 (output swath width) plus 475 (range pulse length) or 4,475. The difference between 4,475 and 5120 is filled with zeros.

The minimum permissable azimuth FFT is a 1751 point transform. However as explained in item 1.3.1 such a transform would yield only one output point. In order to provide more efficient processing it is necessary to batch the generation of many output image points together. As shown in item 1.3.1 the length selected as an optimum compromise between processing and memory requirements is a 3520 point transform. The output points from the azimuth FFT are broken into four 880 point single look groups. Four inverse azimuth FFT transforms yield 440 image points for each of the four looks. The azimuth processing then steps ahead 1760 input azimuth values and the process repeats. The details of the various operations discussed above are provided in the remaining items of this section.

2.2 Processing Flow

The SAR data samples are processed in groups with the returns from 1760 radar pulses in one group. The array unit of the MPP alternates between range processing and azimuth processing. Range processing is performed on the returns from 1760 pulses and then azimuth processing is performed and then range processing is performed again, etc.

Azimuth processing treats the returns from 3520 pulses at one time. Staging memory 2 holds the range-processed returns from two groups of 1760 pulses each. Range processing replaces the oldest group of returns with the range-processed returns from the next group of 1760 pulses. Then azimuth processing occurs on the two groups. In this way, the processing marches along the flight path, 1760 pulses at a time.

Initially, range processing is performed on the returns from 3520 pulses so the first azimuth processing step sees a full set of returns in staging memory 2.

2.3 SAR Data Input

We assume the radar return is sampled N times for each radar pulse where N is 5120 or less. Each sample is a complex number with a 6-bit real part and a 6-bit imaginary part. Each sample uses two adjacent bytes with the imaginary byte following the real byte. Each 8-bit byte contains six bits of data and two fill bits. The N samples for one radar pulse are ordered with the nearest range

first and the farthest range last. The samples for one pulse are followed by the samples for the next pulse and so on.

The samples are first stored in Staging Memory 1. The storage format is chosen to facilitate the transfer of the data to the array unit for range processing. Sixteen adjacent samples are grouped together into four adjacent main stager words (each main stager word holds 64 data bits). The first word of the group holds the low-order 4 real bits of the 16 samples; the next word holds the high-order 4 real bits; the third word holds the low-order 4 imaginary bits; and the fourth word holds the high-order 4 imaginary bits.

The N samples for one radar pulse are stored in N/16 adjacent groups in the main stager (if N is not a multiple of 16 then the last group is only partially filled). The samples for one radar pulse occupy P/4 main stager words where P is a multiple of 16 equal to N, N+1, ..., or N+15.

A one-word gap is inserted after the last group of samples for one pulse and then the samples for the next radar pulse are stored. Thus, the samples for each pulse take P/4 + 1 main stager words. Since P is a multiple of 16, P/4 + 1 is an odd number. This arrangement allows corresponding parts of 32 successive pulses to be read out in parallel for the range processing.

Staging memory 1 can accept data from the input source at rates up to 160 megabytes per second. No processing time is expended in the array unit for this operation.

2.4 Range Processing

2.4.1 Input from Staging Memory

Range processing treats the samples from 32 successive radar pulses at one time. The array unit is divided into 32 sub-arrays with each 32-column by 16-row sub-array treating the samples from one pulse. Let I = 0, 32, 64, or 96 and let J = 0, 16, 32, 48, 64, 80, 96, or 112. The PE's in the sub-array using columns I through I+31 and rows J through J+15 treat samples from pulse J/4 + I/32 of the pulse group.

The 512 PE's within each sub-array treat 5120 samples. If the number of samples per pulse (N) is less than 5120 then the extra slots are filled with complex zeroes. Each PE has 10 slots in its random-access memory for 10 complex-valued samples. The slots are indexed from 0 to

9. The layout of samples within each sub-array is scrambled a certain way to facilitate performing a 5120-point FFT. Let L be the index of a sample (L = 0, 1, ..., 5119). Let A = 4L modulo 5. Let B = 205L modulo 1024. Then L = (5B + 1024A) modulo 5120. If B < 512 then sample L is stored in slot A of PE B. If B > 511 then sample L is stored in slot A+5 of PE B-512. The 512 PE's within each sub-array are numbered with PE's 0 through 15 occupying the first column of the sub-array, PE's 16 through 31 occupying the second column of the sub-array, etc.

The input of data into the array unit for range processing has two steps: (1) the samples are read from the staging memory and put into the correct PE's but not necessarily into the correct slots within the PE's; (2) the samples within each PE are re-arranged using the PE shift registers so they are in the correct slots.

Let C be the integer quotient when L is divided by 512 and let D be the remainder of the division. Then L = 512C + D where C = 0, 1, ..., 9 and D = 0, 1, ..., 511. Note that D = L modulo 512. Since B = 205L modulo 1024 then B = 205D modulo 512 when B < 512; and B = (205D modulo 512) + 512 when B > 511. Thus, sample L should be stored in PE (205D modulo 512) of the sub-array. In the first step of the data input, sample L is stored in slot C of PE (205D modulo 512) of the sub-array.

The second step of the data input re-arranges the samples within each PE to put them in the correct slots. Consider the samples loaded into slot 0 of the PE's by the first step. For L from 0 to 511, sample L is put in slot 0 of PE (205L modulo 512). Thus, for K from 0 to 511, slot 0 of PE K gets sample (5K modulo 512).

We partition the set of PE's within a sub-array into five classes. For K from 0 to 102, slot 0 of PE K gets sample 5K; for K from 103 to 204, slot 0 of PE K gets sample 5K-512; for K from 205 to 307, slot 0 of PE K gets sample 5K-1024; for K from 308 to 409, slot 0 of PE K gets sample 5K-1536; and for K from 410 to 511, slot 0 of PE K gets sample 5K-2048. For G = 0, 1, 2, 3, and 4; we put PE K into class G if slot 0 of PE K gets sample 5K-512G.

If L = 5K-512G then A = 4L modulo 5 = 2G modulo 5, and B = 205L modulo 1024 = K + 512G modulo 1024. For G = 1 and 3; B > 511 and the sample in slot 0 of PE K should be put in slot A+5 = (2G modulo 5) + 5. For G = 0, 2, and 4; B < 512 and the sample in slot 0 of PE K should be put into

slot A = (2G modulo 5). Thus, for G = 0, 1, 2, 3, and 4; the sample in slot 0 of all PE's in class G should be put into slot 0, 7, 4, 6, and 3; respectively.

We have seen how to move the samples in slot 0 of the PE's. Now we examine the samples in the other slots. Note that if the first step of the input process puts sample L into slot C of PE K, then it puts sample L+512 into slot C+1 of the same PE. If A = 4L modulo 5, then 4(L+512) modulo 5 = A+3 modulo 5. If B = 205L modulo 1024, then 205(L+512) modulo 1024 = B + 512 modulo 1024. If B < 512 then (B+512 modulo 1024) > 511 and vice-versa. Thus, if sample L in slot C of a PE is moved to slot E, then sample L+512 in slot C+1 of the same PE is moved to slot F where E and F are related as follows:

E = 0 1 2 3 4 5 6 7 8 9

F = 8 9 5 6 7 3 4 0 1 2

This relation can be used repeatedly to find the correct slot of every sample in the PE's. Table I shows the correct slot for every sample. The correct slot depends on the class (G) containing the PE and the position of the sample after the first step (C).

TABLE I - Correct Sample Slots

CLASS (G)	RANGE OF PE INDICES	SAMPLE POSITION AFTER FIRST STEP (C) 0	1	2	3	4	5	6	7	8	9
0	0-102	0	8	1	9	2	5	3	6	4	7
1	103-204	7	0	8	1	9	2	5	3	6	4
2	205-307	4	7	0	8	1	9	2	5	3	6
3	308-409	6	4	7	0	8	1	9	2	5	3
4	410-511	3	6	4	7	0	8	1	9	2	5

The first step of the data input is performed by the I/O control unit in conjunction with the staging memory. No PE processing cycles are used except for the interruptions when bit-planes are transferred from the S-registers to the random access memories. Since there are 10 samples per PE and 16 bits per sample there are 160 bit-planes to be loaded and 160 processing cycles used for the transfers.

The second step of the data input uses the PE shift

registers to move samples according to Table I. The shift register lengths are set to 18 so 20-bit operands can be shifted end-around through the shift registers and registers A and B. Two data bits from each slot are loaded into the shift registers and using masked-shifts the registers of PE's in class G are shifted 2G times. Then the shift registers are unloaded with two data bits going to each of the 10 slots. The process is repeated 8 times to treat all 16 bits of the samples. It requires 8 x 48 = 384 cycles to do the second step.

Thus, 160 + 384 = 544 processing cycles or 55 microseconds of PE time are required to input the data for range processing. The I/O control unit moves data at a 160 megabyte per second rate so it requires 2 milliseconds to input the data.

2.4.2 Fast Fourier Transform (FFT)

The samples from 32 radar pulses are treated as a group. While group G is being processed in the array unit by the PE's, group G+1 is being loaded into an I/O buffer region of the array unit memory and group G-1 of range-processed samples are being unloaded. The I/O buffer region is 240 bit planes deep so each PE can hold ten processed samples with 12-bit real parts and 12-bit imaginary parts. The input samples have only 8-bit real and imaginary parts and share 160 bit planes of the I/O buffer region with the output samples.

Processing of the samples takes place in a work region of the array unit memory which is 320 bit planes deep (see Figure 4). Input data is moved from the I/O buffer to the work region by step 2 of the input process (see item 2.4.1). The work region is divided into ten 32-bit fields indexed from 0 to 9. As described in item 2.4.1, the sample with index values A and B is stored in field A if $B < 512$ or in field A+5 if $B > 511$.

The 5120-point FFT's are performed in two major phases. In the first phase, five 1024-point FFT's are performed for each 5120-point FFT. Each 1024-point FFT treats samples with a given A index. In the second phase, 1024 five-point FFT's are performed for each 5120-point FFT. Each 5-point FFT treats samples with a given B index. Appendix B of this report describes the process. Note that in the appendix the sample indices are Greek letters with subscripts. Index alpha-sub-1 corresponds to index B and index alpha-sub-2 corresponds to index A.

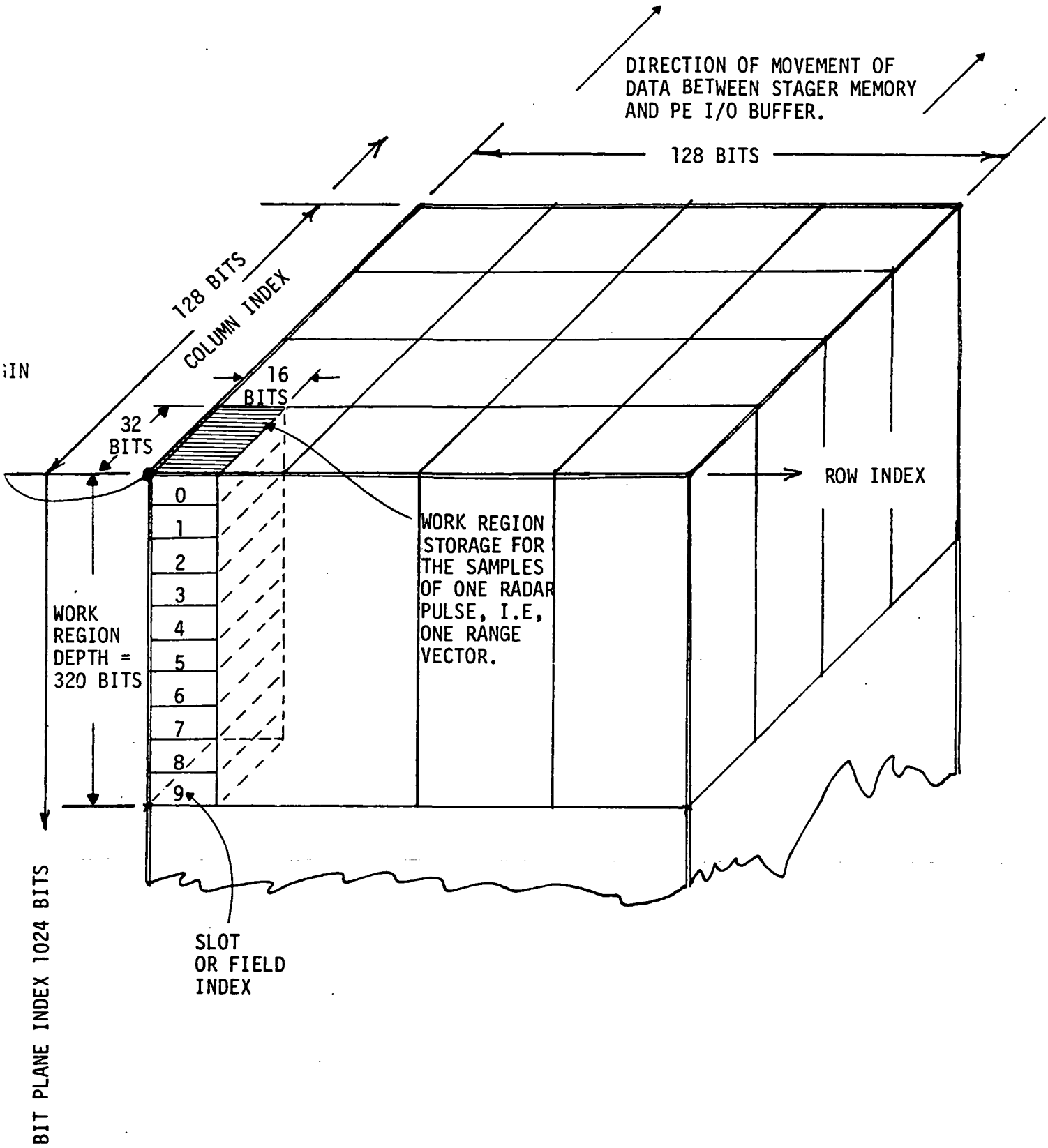

Figure 4 - LAYOUT OF RANGE VECTOR SAMPLES IN WORK REGION OF ARRAY MEMORY

GOODYEAR AEROSPACE

First Phase

Since all five 1024-point transforms are performed in a like manner we only describe the transform corresponding to A = 0. Only work region fields 0 and 5 are involved. Figure 5 shows the layout of the B indices in field 0 and Figure 6 shows the layout of the B indices in field 5. A standard radix-2 FFT is executed. Before each of the last 9 iterations of the FFT the samples are moved between fields 0 and 5 so that all complex multiply operations involve field 5 and not field 0.

The move time for iteration I is given by:

$$TIME(I) = (IMI2(I) + IN + 1) * 2 * (EXDIST + 3) * DIR$$

where

IMI2(I) = integer part of (I-1)/2;
IN = number of bits in real or imaginary part of input sample (6 is assumed);
EXDIST = move distance in number of PE's; and
DIR = number of directions of move (1 if all moves are in same direction or 2 if moves are in both directions).

The above equation assumes that the precision of the real and imaginary parts grows by one bit every other iteration. The elements of the move time are shown in Table II.

TABLE II - Elements of Move Time

I	IMI2+IN+1	EXDIST(I)+3	Product
1	7	19	133
2	8	11	88
3	8	7	56
4	9	5	45
5	9	4	36
6	10	11	110
7	10	7	70
8	11	5	55
9	11	4	44
10	-	-	-
		TOTAL(cycles)	637

The total of Table II is multiplied by 2 and by DIR = 2

31																
	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463
	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

COLUMN INDEX

0 ROW INDEX 15

Figure 5 - B INDICES - FIELD 0

GOODYEAR AEROSPACE

31																
	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007
	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991
	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975
	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959
	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943
	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927
	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911
	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895
	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879
	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863
	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847
	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831
	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815
	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783
	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767
	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751
	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735
	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719
	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703
	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687
	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671
	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655
	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639
	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607
	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591
	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559
	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543
0	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527

COLUMN INDEX

0 ROW INDEX 15

Figure 6 - B INDICES - FIELD 5

to get 2548 cycles or 254.8 microseconds for the move time of the A=0 FFT. The total move time for all five 1024-point FFT's is 1.274 milliseconds.

On each iteration an FFT butterfly is executed between fields 0 and 5. For I = 4, 5, ..., 10; the butterfly time for iteration I is given by:

```
BFLYTIME(I) = (4 * (P + 1) * IMI2) + (8 * I2)  +
              (4 * (P + 3) * (IN + 1)) + (4 * G)
              + (6 * OVR) + 32              cycles
```

where

P = weight-precision factor (equals 10 when the weight has 8-bit real and imaginary parts);
IMI2 = integer part of (I-1)/2;
I2 = integer part of I/2;
IN = input data precision (6 bits);
G = number of guard bits saved during internal multiply (equals 3); and
OVR = add overhead cycles (equals 2).

The sum of the butterfly times for I = 4, 5, ..., 10 equals 3960 cycles.

The butterflies for the first three iterations are simpler because the weight factor angles are multiples of 45 degrees. The butterfly times for iterations 1, 2, and 3 are given by:

```
BFLYTIME(1) =   ((IN + 1) * 3 + OVR) * 4

BFLYTIME(2) =   ((IN + 1) * 3 + OVR) * 4
              + ((IN + 2) * 3 + OVR) * 4

BFLYTIME(3) =   ((IN + 3) * 3 + OVR) * 6
                    + (P * (IN + 3) + 8)
                    + (IN + 4) * 3 * 2 +
                ((IN + 2) * 3 + OVR) * 4
```

With IN = 6, P = 10, and OVR = 2, the sum of the butterfly times for the first three iterations is 724 cycles. Thus, the butterfly time for the ten iterations of the A=0 FFT is 724 + 3960 = 4684 cycles. The butterfly time for the five 1024-point FFT's is 5 * 4684 = 23420 cycles or 2.342 milliseconds.

The time to execute the first phase is the move time plus the butterfly time, 1.274 + 2.342 = 3.616 milliseconds.

Note that the samples from 32 radar pulses are treated simultaneously so in reality the array unit performs 160 1024-point transforms in this time period. The real and imaginary parts of the input samples are each six bits (5 bits plus sign). The output samples have 12-bit real and imaginary parts (11 bits plus sign).

Second Phase

To complete the 5120-point FFT for each radar pulse, 1024 five-point transforms are performed. Each five-point transform combines samples for a given index B. Each PE contains the samples for two values of B so no inter-PE moves are required. A five-point transform is executed as outlined in Appendix C. Figure C-4 of the appendix shows a flow chart. Table III establishes the execution time.

TABLE III - Five-Point FFT Execution Time

Scalar Mult.	Adds	Precision	Number of Cycles
	8	12	8*(12*3+2) = 304
	6	13	6*(13*3+2) = 246
8		13*14	8*97 = 776
	2	13	2*(13*3+2) = 82
	8	13	8*(13*3+2) = 328
	2	14	2*(14*3+2) = 88
	8	14	8*(14*3+2) = 352
			TOTAL =2176

Each PE performs two five-point transforms so the time for phase 2 is 2 * 2176 = 4352 cycles or 0.4352 milliseconds. Note that a total of 32768 five-point transforms are performed in this time period.

Timing Summary

The time required to execute the forward range FFT for 32 radar pulses is the sum of the phase 1 and phase 2 times: 3.616 + 0.4352 = 4.052 milliseconds.

2.4.3 Frequency Domain Processing

After the forward range FFT, frequency domain processing to achieve motion compensation, partial range walk correction, and range compression commences.

In the frequency domain, the former two corrections force processing that causes the frequency domain range vector data to be multiplied by a complex number with unit magnitude. The compression operation requires the equivalent of a complex multiply using weights whose magnitudes are nearly unity. The three complex correction factors are combined prior to complex multiplying the FFT'd range data.

Motion Compensation

Motion compensation must account for the fact that flight perturbations displace the true origin of the range vector indices from the assumed origin. The assumed origin lies along the unperturbed path of the radar platform; the true origin will be off the path. Generally, only the component of the shift in the direction of the radar beam center is significant. The distance shift for each radar pulse is reduced to a corresponding index shift value by the host computer that supports the MPP. For the ERSAR mission, the shift value, in range sample index units, is likely to be fractional. This value is combined with the range walk shift for a radar pulse and then scaled prior to utilization by the MPP.

Range Walk

When the radar is used in a squint mode, a plot of the power returned from a point target in a range sample index/pulse index coordinate system is found to be an arc that lasts for about 7000 pulses. The chord of the arc is canted relative to the radar pulse index axis; the cant becomes more severe as the squint angle increases. For the ERSAR L case, a cant of as much as 86 range sample indices in 7000 pulses can result. (When the squint angle is zero, no canting results. The chord then is parallel to the radar pulse axis.)

It is found that for processing convenience, it is useful to convert to a sequence of local coordinate systems whose sample index axes lie perpendicular to the chord and whose pulse index axes lie parallel to the chord. The origins of the rotated coordinate systems are set to lie on the radar pulse index axis of the radar pulse index/range sample index coordinate system.

By slipping (range walking) the indices of the samples of successive pulses by ever increasing amounts, the effect of rotation can be accomplished. Every 1760 radar pulses,

the origin of the rotated coordinate system is re-initialized, and the linear increase of slippage as a function of pulse sample index repeats as before.

The range walk delay value for a pulse is developed by the host. When most severe, the index shift for the pulse with index P is given by:

$$OFF=(86/(4*1760))*(P \text{ modulo } 1760).$$

The range walk index shift is combined with the index shift developed for motion compensation. The residual of the shift is treated in the frequency domain as follows. The largest integer multiple of 16 that can be subtracted from OFF without causing a negative result is subtracted from OFF to get the residual shift value. This value is scaled with the factor 1/(Range FFT size). The resultant value (described as a 12 bit integer of the form (1.4.7)*(2**(-12)) is called CORR(P). ((x.y.z) denotes x sign bits, y integer bits, and z fractional bits in a x + y + z bit value.)

Data index shifts that are integer multiples of 16 are handled after the range inverse FFT.

Use of CORR(P)

In the frequency domain, data shifting is accomplished by multiplying all frequency domain samples by:

$$C(N) = A(N) * (COS(B(N)) + i * SIN(B(N)))$$

where N is the index of a frequency domain sample, B(N) is the correction phase angle, and A(N) is the correction magnitude. Data that describe the frequency domain representation of the range reference function in a magnitude/phase form, namely, (A(N),REFANG(N)), in conjunction with CORR(P), totally defines C(N). In particular, in units of rotations:

$$B(N) = (N * CORR(P))/4096 + REFANG(N).$$

To determine C(N), the host sends CORR(P) values to the MPP. The MPP performs the subsequent operations needed to find C(N).

The quantity, N*(CORR(P)) is first computed. Since the FFT's of 32 pulses exist within the array, it is necessary to move 32 different CORR(P) values to the MPP scalar register and to write them (masked) to the PE registers using 32 P-region masks resident in the arrays. When all the PE registers are loaded, the register values

are loaded into a 12-bit wide CORR field. The execution time required for loading is 400 cycles. (Since 10 fields for samples exist in the arrays, 10 fields for frequency indices must also exist. These fields are 13 bits each.)

To develop N*CORR(P), 10 field multiplies are required. The computation time is (14*14+12)*10=2080 cycles. the net angle, in rotations, is found by putting the radix point (12+7) bits to the left of the right edge of the LSB of the product field. The integer part of the product is discarded.

Composite Phase Angle

The phase angle of the FFT'd range reference function is then added to the combined motion and range walk phase angle. Each such angle is described as a 6 bit integer of the form (0.0.6). The reference function's magnitude is described as a 6-bit unsigned scaled integer. (Note that 12*10=120 bit planes hold the reference function data.) Thus, the add time = 230 cycles.

Cartesian Form of Weight

Next, the rotation angle, B(N), is used to develop the complex value:

$$COS(B(N)) + i*SIN(B(N)).$$

Both COS(B(N)) and SIN(B(N)) are developed for the 10 fields. Using a second order polynominal technique, the complex components corresponding to one field can be computed in less than 250 cycles. Thus, the computation time for 10 fields will be less than 2500 cycles.

Using the magnitude of the frequency domain range reference function, A(N), the final complex correction weights can be accomplished with 20 field multiplies. The time for such multiplies will be about 880 cycles.

Complex Multiply

Having developed the complex correction weights as (8,8) values, the final complex multiplication of the frequency domain range vector data can be accomplished using 10 complex field multiplies (or 40 real multiplies and 20 field adds.). Assuming (1.0.13,1.0.13) data and (1.0.7,1.0.7) weights, the total time for the 10 complex field multiplies is 6980 cycles.

Total Time

The total time for motion compensation, partial range walk correction, and frequency domain correlation follows:

Motion Comp/Range Walk Data Loading	=	.040 millisec
Phase Angle Calculation	=	.208 millisec
Summing of Phase Angles	=	.023 millisec
Generation of COS, SIN of Phase	=	.250 millisec
Reference Magnitude Multiply	=	.088 millisec
Complex Multiply	=	.698 millisec
TOTAL	=	1.307 millisec

2.4.4 Inverse FFT

After frequency domain filtering, motion compensation, and fractional range walk correction, the samples are transformed back to the time domain with an inverse FFT. The samples for each pulse are first block normalized and the block normalization factors are stored in the main control memory of the MPP. The time required for block normalization is 0.5 milliseconds. The inverse FFT is performed by reversing the steps of the forward range FFT.

Inverse 5-point FFT

Inverse five-point transforms are performed following the steps in Figure C-4 of Appendix C from right to left. The add/subtract operations of Table III are carried out with 16-bit precision and the scalar multiplies are carried out using 14-bit scalar reciprocals. When appropriate, the results are scaled to prevent overflow and underflow.

Each PE performs an inverse 5-point FFT in 2660 cycles. Two transforms per PE require 5320 cycles or 0.532 milliseconds. Note that the array unit performs 32768 transforms in this time period. After the inverse 5-point transforms a second block normalization is performed requiring 300 microseconds.

Inverse 1024-point FFT

The steps of the forward 1024-point FFT's (item 2.4.2) are reversed to perform the inverse 1024-point FFT's. The iteration coefficients require no change in location.

The steps are executed with 16-bit real and imaginary parts. After the fifth and tenth iterations normalization steps are added. The last normalization step leaves the data with 12-bit real and imaginary parts.

With D-bit data operands and W-bit weight operands the number of cycles to perform the 4 real multiplies and 6 real add/subtract operations of a butterfly is given by:

$$BFLYTIME = 6 * (2 * D + 2) + 4 * (P(W) * D + K(W)).$$

Assuming D = 16 and W = 8, P(W) = 10 and K(W) = 8 so BFLYTIME = 876 cycles. Five butterflies are required (one for each value of index A). The BFLYTIME expression is good for the iterations corresponding to iterations 4 through 10 of the forward FFT. The total butterfly time for these seven iterations is 7 * 5 * 876 = 30,660 cycles.

The butterfly time for the iterations corresponding to iterations 1, 2, and 3 of the forward FFT is computed using the equations in item 2.4.2 for BFLYTIME(1), BFLYTIME(2), and BFLYTIME(3) with (IN + 1), (IN + 2), (IN + 3), and (IN + 4) replaced by 16 and OVR = 2 and P = 10. The butterfly time for these three iterations is 5 * 1364 = 6820 cycles.

Thus, the total butterfly time for the ten iterations of the 1024-point inverse transforms is 30,660 + 6820 = 37,480 cycles or 3.748 milliseconds.

The move time between successive iterations is readily computed. The length of the moves after iterations 1 through 9 are, 16, 8, 4, 2, 1, 8, 4, 2, and 1, respectively. The total length of the moves is 46 so 46 route cycles are required per bit plane. Each bit plane is accessed 9 times and each access uses three cycles (read, read masked and store) so there are 27 cycles per bit plane for accessing. The number of cycles per bit plane is doubled since moves occur in both directions so each bit plane requires 2 * (46 + 27) = 146 cycles. A total of 160 bit planes are moved so the move time is 160 * 146 = 23660 cycles or 2.336 milliseconds.

Timing Summary

The time to invert the 32 5120-point vectors back to the time domain is summarized below:

First Normalization	= 0.500 milliseconds
Inverse 5-point FFT's	= 0.532
Second Normalization	= 0.300
Third and fourth Normalizations	= 0.800
Inverse 1024-point FFT Butterflies	= 3.748
Inverse 1024-point FFT Move Time	= 2.336
TOTAL	= 8.216 milliseconds

2.4.5 Output to Staging Memory

The range-processed samples are sent back to staging memory 2 to be collected and then re-read in corner-turned fashion for azimuth processing. The output process is much like the input process (see item 2.4.1) except that the real and imaginary parts of each sample are 12 bits long instead of 8 bits and the group of samples for the 32 pulses are moved a multiple of 16 range bins to make any gross range-walk correction required. First the samples in the ten slots within each PE are rearranged to invert the permutation performed by the second step of the input process. Then the samples are output to staging memory 2 which inverts the permutation performed by the first step of the input process and shifts the location of the data for the gross range-walk correction.

Rearrangement of the slots within each PE requires 576 processing cycles. Outputting of the data interrupts the processing 240 times (once per bit-plane) so a total of 816 processing cycles (82 microseconds) are required. The I/O control unit moves data at 160 megabytes per second so it requires 3 milliseconds to output the data.

2.5 Azimuth Processing

2.5.1 Input from Staging Memory

Azimuth processing reads 32 range-processed samples from each of 4096 pulses into the array unit. The layout of data in the array unit is selected to facilitate performing an FFT in the azimuth direction, interpolating across range bins to correct for range curvature, dividing the set of data into four looks, and then performing an inverse FFT within each look. The array unit is divided up into 32 sub-arrays with each sub-array covering 4 rows and 128 columns. Each PE holds eight

range-processed samples and each sample has a 12-bit real part and a 12-bit imaginary part.

For I an integer from 0 through 31, range-bin I of the 32 range-bin group is treated in the sub-array covering rows 4I, 4I+1, 4I+2, and 4I+3 of the array unit. Each PE has eight slots indexed 0 through 7. Slot J of PE K of the sub-array holds data from pulse 512J+K of the 4096 pulses being treated. The PE's within each sub-array are arranged as follows. Let K = 16W + 4X + 2Y + Z where W = 0, 1, ..., or 31; X = 0, 1, 2, or 3; Y = 0 or 1; and Z = 0 or 1. Processing element K is located in row X and column 64Z + 32Y + W of the sub-array.

Range curvature correction may slide the range bins up to seven places. This fact coupled with the usage of a four-point interpolation formula means that out of the 32 range bins being processed only 23 bins may have useful output data. Thus, successive azimuth processing steps want some redundancy in the input data. Each step wants 32 range bins but the first nine bins should be a repeat of the last nine bins of the previous step. In the staging memory each main stager word holds data from 16 successive range bins. The data for the 32 bins required by the azimuth processing may overlap 3 main stager words. Thus, the output sub-stager of the staging memory reads a group of 48 bins from the main stager and then sends only the desired 32 bins to the array unit. This means that the basic I/O rate of the staging memory is reduced to only 2/3 of 160 megabytes per second (106.67 megabytes per second).

Each of the eight slots has 24 bit-planes so there are 192 bit-planes to be loaded into the array unit. This will interrupt array unit processing 192 times for a loss of 192 processing cycles (19.2 microseconds). The 393,216 bytes of an array load will be transferred by the I/O control unit in 3.69 milliseconds.

2.5.2 Fast Fourier Transform (FFT)

The azimuth FFT processing proceeds in a manner similar to the range FFT processing. As in range processing, a total of 32 azimuth FFT's are performed concurrently. The 4096-point FFT is about the same as the 5120-point FFT so one might expect a similar data layout in the array unit. But the range curvature correction requires data moves between FFT's so the layout is different.

Layout

The PE allocation for one 4096-point azimuth FFT uses 4 rows by 128 columns (figure 7). Each PE holds eight complex values. Each value is allocated 36 bits (18 real and 18 imaginary). When first loaded with input vector data only 24 bits per value are used (12 real and 12 imaginary). The remaining bits of each value are used to increase dynamic range as processing progresses. The I/O buffer is the same size as the work region, 8 x 36 = 288 bit planes. A total of eight 12-bit array fields is used to store output look magnitudes and the remaining storage is used for storing output look complex values. Figure 8 illustrates the layout of PE's within a typical sub-array.

Flow

As in the range processing case, group G+1 of 32 azimuth vectors is loaded into the I/O buffer as group G-1 of 32 processed vectors is being unloaded from the I/O buffer. Meanwhile, the PE's are processing group G in the work region.

Processing Time

The first three iterations of the forward azimuth FFT are treated specially. The remaining iterations are treated in a standard radix-2 butterfly manner except for precision increase. The formula for the execution cycles for all iterations is shown below:

Iteration (I)	Number of Cycles
1	NFP*(4*(IOP(I)*3+OVR))
2	NFP*((2*IOP(I)+IIP(I))*6+5*OVR)
3	NFP*2*(((2+S*P(W)/3)*IIP(I)+IOP(I)+1)*3 +S*K(W)+2.5*OVR)
(4-12)	NFP*4*((P(W)+1)*IIP(I)+2*IOP(I)+K(W)+G+1.5*OVR)

where:

NFP = the number of array unit field pairs;
OVR = the overhead set-up charge;
I = iteration index;
IOP(I) = number of bits in real or imaginary part output from iteration I;

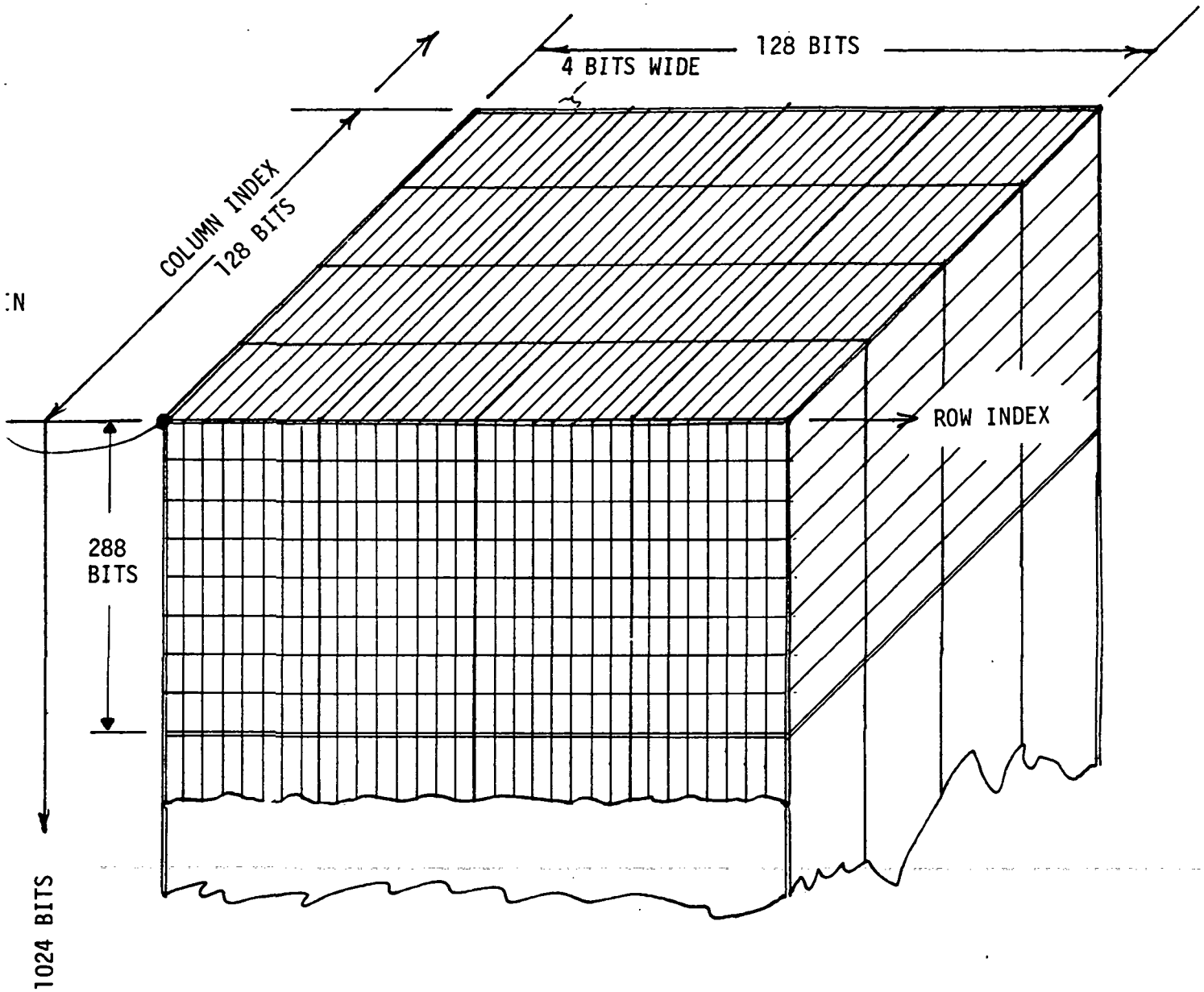

Figure 7 - FORWARD AZIMUTH FFT DATA LAYOUT
IN WORK REGION

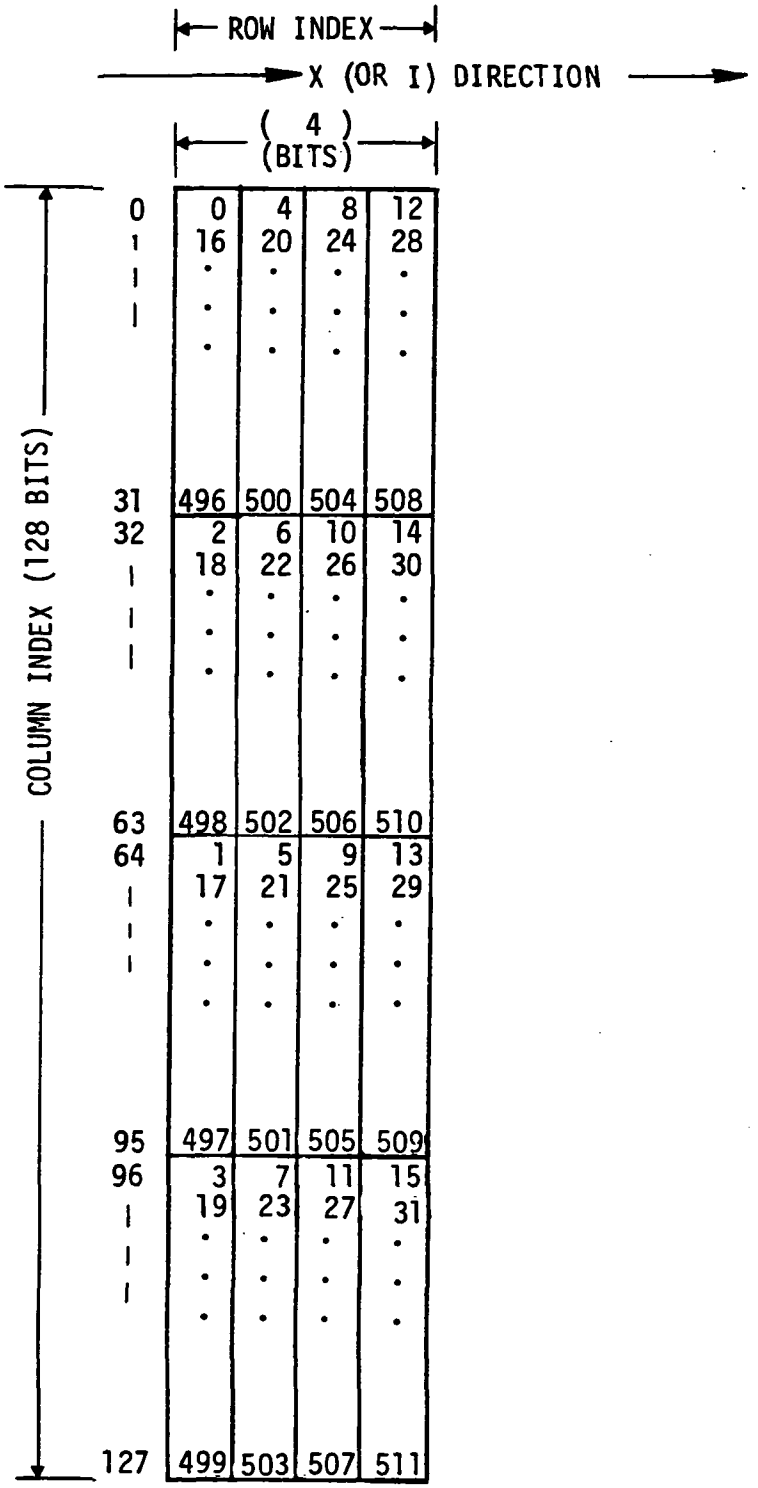

Figure 8 - TYPICAL SUBARRAY

IIP(I) = number of bits in real or imaginary part input to iteration I;
S = scalar multiply enhancement factor accounting for the higher speed of scalar multiplies over field-by-field multiplies;
P(W) = multiplicative factor for a W-bit coefficient;
K(W) = additive factor for a W-bit coefficient;
G = number of guard bits used in add/subtract operations during complex multiplies.

For the case at hand, NFP = 4, OVR = 2, S = 0.5, W = 8, P(W) = 10 (see Table IV), K(W) = 8 (see Table IV), and G = 3.

TABLE IV - Multiplication Factors

W	P(W)	K(W)	W	P(W)	K(W)
3	6	2	18	22	16
4	6	4	19	22	18
5	10	2	20	22	20
6	10	4	21	26	18
7	10	6	22	26	20
8	10	8	23	26	22
9	14	6	24	26	24
10	14	8	25	30	22
11	14	10	26	30	24
12	14	12	27	30	26
13	18	10	28	30	28
14	18	12	29	34	26
15	18	14	30	34	28
16	18	16	31	34	30
17	22	14	32	34	32

To find IIP(I) and IOP(I) for a given iteration index, I, use Table V. Using these values, the processing time for 32 4096-point forward azimuth FFT's is 31992 cycles or 3.199 milliseconds.

TABLE V - Forward Azimuth FFT Parameters

Iteration Index I	Lateral Shift (steps/plane) STEP(I)	Input Precision IIP(I)	Output Precision IOP(I)
1	-	11	12
2	-	12	13
3	16	13	13
4	8	13	14
5	4	14	14
6	2	14	15
7	1	15	15
8	2	15	16
9	1	16	16
10	32	16	17
11	64	17	17
12	-	17	18

Data Exchange

The execution time for the data exchanges of an iteration is given by:

MOVE TIME = IOP(I)*(STEP(I)+3)*NFP*4 cycles

From Table V the total move time for all iterations is 4.019 milliseconds.

Timing Summary

The execution time for processing a group of thirty-two 4096 azimuth element vectors through a forward 4096-point FFT is summarized below:

Processing Time = 3.199 milliseconds
Data Exchange Time = 4.019

TOTAL = 7.218 milliseconds

2.5.3 Frequency Domain Processing

As an initial step to curvature correction, the data of the 8 fields must be exchanged in two power of 2 lateral moves so that all data associated with a given look is in association with only 4*32 PE's. The shift time is computed using the move time expression of item 2.5.2 with IOP(I)=18, NFP=4, and STEP(I)=64+32. Thus, the look

associate move time = 28512 cycles.

Normalization

After the data has been associated by look frequency regions, the data is block normalized over all 32 range indices. The normalization value is first generated by look index and then the normalization procedure commences. By restricting normalization to 15 or less bit positions, it is possible to perform the normalization of the data of all 8 fields of 36 bit planes/field in less than .075 milliseconds.

Interpolation

Range curvature correction commences next. To perform this task, azimuth frequency domain data must be slipped in the range index direction up to 7 index positions. Since the actual shift generally involves a fractional component, a cubic convolution technique, which uses the samples with range indices closest to the desired index, is used to approximate the interpolated value. In particular, the data value at azimuth frequency domain index a and range index r is to be the sample value at a,r+1+x where $0 \leq x < 7$. The shift value x consists of an integer N from 0 to 6 plus a fraction f. Thus, to generate the interpolated sample value, the value of f and the 4 sample values with indices a,r+N+J where J=0, 1, 2, 3 are required to establish the sample value at a,r.

Since N and f can be precomputed for each a,r (based on geometry and mission parameters) and can be associated with the PE that corresponds to the a,r point, a particular PE knows from where it must get its data (N) and it has the correct parameter (f) to develop the interpolated value for a,r. In fact, an f-field is loaded for each of the 8 fields of sample data in the array. (Since f is defined as a 3-bit integer, the 8 fields require 24 bits. The fields are loaded after the sample data is loaded.) Also, instead of loading 8 a-fields into the array, 8 mask sets corresponding to each a-value are loaded into the array. Each mask set of 8 bits is used to strip off required data as it is shifted across the PE plane in an increasing N direction.

A total of 8 x 4 = 32 lateral shifts of each data plane is required to make the data moves and to strip off the desired data. Each plane must be read once and written four times. Thus, (36 x 8) x (32 + 1 + 4) = 10656 cycles

are required to associate the data with the appropriate PE's (i.e., (a,r) locations).

After the data are at location, the sample at (a,r), SAMP, is computed using the cubic convolution interpolation expression:

$$SAMP = I1+f*(I2-I0+f*(I0-I1+(f-1)*(I1-I0+I3-I2))).$$

Real and imaginary fields are treated separately so SAMP must be computed for 16 half-fields, each of which is 18 bits. Each computation requires 672 cycles, and so, a total of 10,752 cycles is required for interpolation processing.

Correlation

Once the data are interpolated, it can be frequency-domain cross correlated against each of the four look reference patterns. The frequency domain versions of the reference patterns are stored within the arrays for each of the 8 interpolated data fields. Each reference is described by 6 bits of real and 6 bits of imaginary data. It requires 96-bit planes (not 4*96) bit planes for storage.

The execution time required to cross correlate all four looks in the frequency domain is equal to the time of 8 complex field times field multiplies, namely, 8 fields*68 cycles/field=5456 cycles.

When the frequency domain correlation processing is completed, the data can be transformed back into the time domain.

Total Time

The total execution time breakdown for curvature correction and correlation for 32 range indices by 440 output azimuth samples follows:

Look associate move	= 2.851 milliseconds
Normalization	= .075 milliseconds
Interpolation grouping moves	= 1.066 milliseconds
Interpolation	= 1.075 milliseconds
Correlation	= .546 milliseconds
TOTAL	= 5.613 milliseconds

2.5.4 Inverse FFT

After correcting for curvature and multiplying the transformed data by the transformed references for the four looks, the azimuth data corresponding to four looks and 32 range bins exist within the array. Each look corresponds to one quarter of the 4096 azimuth frequency bins; thus, if the 1024 frequency bins of a quarter are inverse transformed, the result is one look in the time domain with a time spacing four times as great as the input spacing. The reduction in sampling rate is consistent with the reduction in signal bandwidth.

Data Layout

The data layout allows the immediate start of the inverse 1024-point FFT. The data for each inverse FFT lie within a 4 row by 32 column PE sub-array. A total of 128 inverse FFT's are performed in parallel. Only 4 x 23 = 92 of the resultant time vectors are useful since the remaining vectors have not been fully corrected for range curvature. The process is repeated 174 times to cover the full swath of 4000 range bins. Each PE contains eight 36-bit complex elements.

Processing Time

The time to process 128 inverse 1024-point FFT's is determined using Tables IV and V and the cycle count expressions of item 2.5.2. All processing is performed on 18-bit real and 18-bit imaginary components. The real and imaginary parts of the iteration weight vector is defined using 8 bits. The processing time is 28065 cycles.

Normalization Time

Block floating point normalization is performed after iterations 2, 6, and 10. The total time for such normalizations is less than 0.800 milliseconds. It should be noted that the last normalization rescales the output to account for a host-specified rescaling constant. Note that since only 440 of the 1024 elements of each output vector are useful and these lie within just 4 of the 8 array fields, the last normalization or re-scaling step requires less processing.

Data Exchange Time

Data moves required between iterations of the inverse FFT are not as extensive as those for the forward FFT's. The lateral shift of each bit plane after each iteration is listed below:

Iteration (I)	STEP(I)
1	1
2	2
3	1
4	2
5	4
6	8
7	16
TOTAL	34

No lateral shifts are required after iterations 8, 9, and 10 because the data exchanges occur between array fields within the PE's. The total time for data exchange is computed using the data exchange cycle count expression of item 2.5.2. The total time is 15840 cycles.

Timing Summary

The total time required to inverse FFT the 128 1024-point vectors is shown below:

Processing Time	= 2.806 milliseconds
Data Exchange Time	= 1.584
Normalization Time	= 0.800
TOTAL	= 5.190 milliseconds

2.5.5 Output to Staging Memory

After azimuth processing the complex data for one look may be output to a display. The data format in the array unit is as follows. Look 0 is in the first 32 columns of the array unit, look 2 is in the next 32 columns of the array unit, look 1 is in the next 32 columns of the array unit, and look 3 is in the last 32 columns of the array unit. Each look has 512 bins in the azimuth direction and 23 to 32 bins in the range direction. Let J = 128X + 4Y + Z where X = 0, 1, 2, or 3; Y = 0, 1, ..., 31; and Z = 0, 1, 2, or 3. Range bin K (K = 0, 1, ..., 31) of azimuth bin J is stored in slot X of the PE in row 4K + Z

of column Y of the look. Each sample has a 16-bit real part and a 16-bit imaginary part.

To output the complex data for one look 128 bit-planes are transferred to the S-registers but the S-registers are only shifted 32 columns for each bit-plane. The array unit processing is interrupted for 128 processing cycles (13 microseconds) and the I/O control unit takes 410 microseconds.

2.6 Combining Looks

2.6.1 Input from Staging Memory

To combine the four looks an array of accumulated values is kept in staging memory 1. After the samples have been processed in the azimuth direction, their magnitudes are computed and then added to the accumulated values. The array of accumulated values is shifted appropriately so that after four passes the four looks have been added together.

The array of accumulated values has 14-bit operands so that four 12-bit magnitudes can be summed together without overflow. The values are rounded down to 12 bits when sent to the display.

In the array unit the array of accumulated values fills 56 bit-planes (14 planes for each of the four slots). The inputting of the array interrupts the processing 56 times for a loss of 5.6 microseconds of processing. The I/O unit transfers the array in 717 microseconds.

2.6.2 Processing

The magnitudes of the samples are calculated by extracting the square root of the sum of the squares of the real and imaginary parts. This requires 882 processing cycles for each array of samples. A 12-bit magnitude is computed from the 16-bit real and imaginary parts. Combining the four looks requires the addition of the magnitude arrays to the accumulated value arrays. Addition of a 12-bit magnitude array to a 14-bit accumulated value array takes 40 processing cycles. Thus, processing of the four arrays requires (882 + 40) x 4 = 3688 processing cycles (368.8 microseconds).

2.6.3 Output to Staging Memory

As described in item 2.6.1 an array of accumulated values is kept in staging memory 1. After adding in the current set of looks the array is output to the staging memory. The time required is 717 microseconds by the I/O control unit and 5.6 microseconds of processing time.

One quarter of this array (the quarter that has accumulated all four looks) is output to the display from staging memory 1 while the other three quarters are shifted appropriately and fed back to the array unit later for the addition of further looks.

2.7 Clutterlock

The purpose of the clutterlock is to determine the orientation of the synthetic aperture with respect to the physical antenna aperture. Unless the orientation is properly maintained, either by an accurate antenna stabilization system or through correction signals developed from the clutterlock indications, errors will appear in the image. The most apparent effect is that the PRF will be inadequate for sampling the doppler frequencies.

The operation of the clutterlock is based upon the assumption that the magnitude of the targets giving rise to the returns is uniformly distributed throughout the illuminated area. Hence the summation of the positive and negative doppler signals should be zero if the antenna is properly oriented. The extent of the departure is indicated by the magnitude and sign of the clutterlock signal.

The clutterlock could be implemented by summing the positive and negative frequency signal values after the azimuth FFT. A computationally simpler approach is obtained by using the results after the range inverse FFT. At this point the signals themselves represent a summation over the physical aperture.

The clutterlock signal is derived from the 64 range bins at the nearest range and the 64 range bins at the farthest range. In each range bin, the total phase shift is computed over a synthetic aperture (1760 pulses). The total phase shift may include several complete revolutions around the circle. We assume the phase shift between adjacent samples is less than a right angle (if it were greater than a right angle then either the assumption of uniform target distribution is violated or the returns have a very low magnitude). The signs of the real and the imaginary parts are used to identify the quadrant of each sample. The quadrants of adjacent samples are compared to derive the net quadrant changes and the net

quadrant changes are summed over the 1760 samples to get the total phase shift. If two adjacent samples in a range bin are in opposite quadrants then the range bin is discarded because the phase shift between these samples is greater than a right angle.

The average phase shift of the 64 near range bins is computed by summing the total phase shifts of the bins and dividing by the number of bins. The average of the 64 far range bins is computed similarly. The averages are smoothed by passing the results of each synthetic aperture through a low pass filter with a time constant of several aperture lengths. The smoothed averages are used to derive the clutterlock signal. Processing requires 0.4 milliseconds per synthetic aperture length.

2.8 Autofocus

Autofocus is a method by which the image focus can be improved by means of computations made on the images. The method attempts to correct for azimuth parabolic phase error, a major cause of image defocusing. The approach is based upon the fact that images from successive looks at the same target area over a short azimuth region will not show azimuth displacement if there is no phase error. However, if azimuth parabolic phase error exists, the two images will show displacement in the azimuth position of targets. The situation is illustrated in Figure 9 in which parabolic azimuth phase error results in a difference in look angles.

A cross correlation between the two images in the azimuth direction indicates the amount of parabolic azimuth phase error by indicating quantitatively the azimuth geometrical displacement between the two images. If the magnitude of the phase error is changing slowly, the usual condition, a shift of three values in azimuth from the previously determined value should locate the new value of the peak of the cross-correlation curve. The amount of phase error, thus determined can be used to correct the phase error in the images until a new cross-correlation is made. The intervals between cross-correlation depends upon the rate at which the parabolic azimuth phase error is changing and is assumed in this report to be required once for every ten aperture lengths traveled.

For computational convenience autofocus correlates the magnitude of look L to the sum of the magnitudes of looks 1 through L-1 for L = 2, 3, and 4. Multiplying the 12-bit look magnitude by the 14-bit magnitude sum requires 208 cycles. The multiplication is repeated for each of the four array fields in a PE and the products are summed to form a 28-bit sum within each PE. The within-PE operations require 4 * 208 + 140 = 972 cycles.

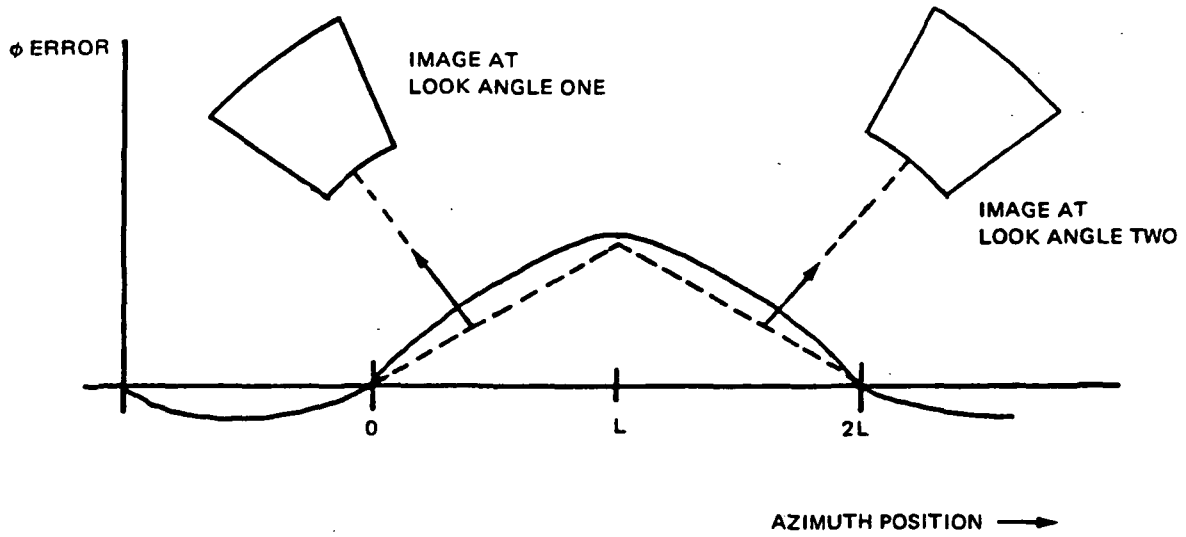

Figure 9 - The Autofocus Principle

Then the sums of the PE's within each 32 x 32 sub-array of the array unit are added together into the corner elements of the sub-arrays. The summation of N-bit operands D places apart to generate (N+1)-bit sums requires:

$$N * (D + 2) + 2 \text{ cycles.}$$

There are ten steps to the sub-array summation where:

N = 28, 29, 30, 31, 32, 33, 34, 35, 36, and 37; and

D = 16, 16, 8, 8, 4, 4, 2, 2, 1, and 1;

respectively. Sub-array summation requires a total of 2541 cycles. The 38-bit sub-array sums are then sent to the array control unit for final summation. This requires 114 cycles.

Thus, the time required is 972 + 2541 + 114 = 3627 cycles to do the correlations for 23 range bins for one synthetic aperture. The process is repeated three times with the looks shifted -1, 0, and 1 place in the azimuth direction. To treat the complete swath of 4000 range bins the steps are repeated 174 times for a time of 174 * 3 * 3627 = 1,893,294 cycles or 190 milliseconds. If the autofocus operation is performed once for every ten aperture lengths the time penalty is 19 milliseconds per aperture.

2.9 Output to Display

Staging memory 1 collects output data for the display. As described in item 2.6.3, one quarter of the array of accumulated values contains the summation of the magnitudes of the four looks. Item 2.5.5 describes the output of the complex values of one look to the staging memory. Staging memory 1 stores the output arrays in azimuth- oriented format so the display should scan each range bin for a full 440 samples before proceeding to the next range bin. The display reads the output data from the staging memory through a buffer so that the output port of staging memory 1 can be time-shared with other processes. The output port transmits data at a 160 megabyte per second rate. No processing cycles in the array unit are required to handle the movement of data from staging memory 1 to the display.

2.10 Timing Summary

The time to range process the samples from 32 radar pulses can be calculated by summing the times shown in items 2.4.1 through 2.4.5. Range processing requires 0.055 + 4.052 + 1.307 + 8.216 + 0.082 = 13.712 milliseconds for 32 radar pulses. Array unit input and output times for 32 radar pulses are 2 and 3 milliseconds, respectively, so when array unit input, output and processing are overlapped the time is governed by processing (13.712 milliseconds). To range process the samples from 1760 pulses requires 55 x 13.712 = 754.2 milliseconds.

Azimuth processing treats 32 input range bins by 3520 pulses at one time, and outputs about 23 range bins by 1760 pulses. Processing time is calculated by summing up the times shown in items 2.5.1 through 2.5.5. Azimuth processing requires 0.019 + 7.218 + 5.613 + 5.190 + 0.013 = 18.053 milliseconds. Array unit I/O time can be overlapped with processing. The process is repeated 174 times to process all 4000 range bins, 174 x 18.053 = 3141 milliseconds.

The time to combine the looks is computed by summing the times shown in items 2.6.1, 2.6.2, and 2.6.3. Processing time is 5.6 + 368.8 + 5.6 = 380 microseconds. For the full swath width of 4000 range bins the process is repeated 174 times for a total of 67 milliseconds.

Item 2.7 shows that the clutterlock requires 0.4 milliseconds per synthetic aperture length of 1760 pulses and item 2.8 shows that the time penalty for autofocus is 19 milliseconds per aperture.

The total processing time for one synthetic aperture length is 754.2 + 3141 + 67 + 0.4 + 19 = 3981.6 milliseconds or a little less than 4 seconds. The bulk of the processing time is spent in azimuth processing where the array unit layout is not optimum for the FFT operations.

SECTION 3 - FLEXIBILITY

Since the MPP is fully programmable in its operation, it can be readily adapted to any desired operating parameters and modes. The range and azimuth compression ratios can be changed as desired. As indicated in section 2 the range and azimuth compression processing is performed with FFT's. Changing these ratios therefore means changing the length of the FFT. There are certain FFT lengths, small multiples of powers of two, which are more convenient to use than other values. If these special FFT lengths do not exactly match a desired length, the next higher special value can be selected and the difference between the desired value and the special value filled with zeros. For example if a 187 point transform were desired, a 192 point (3x64) transform would be selected and 5 zero values filled in. The special values can be pre-determined over a range of values, say 10 to 5120, so that the proper value can be selected by the software.

Various swath widths can also be handled with different software programs. The processing time is nearly proportional to the product of the FFT length and the number of looks. Trade-offs between resolution, number of looks, and swath width can be made with little change in processing time if the product just mentioned is maintained. For example a four look half swath processing mode can be readily converted to a two look full swath processing mode at nearly the same total processing time. If different total processing times are permitted, a full range of resolution, number of looks, and swath widths would be possible.

The MPP design is modular so that it could be expanded to accommodate such items as greater swath width or more parallel looks. The expansion could be performed in several different ways as required for the particular application. The array memory can be expanded which would permit faster array unit operation. Several staging memories can be operated together to provide greater memory capacity. Finally several array units can be operated together with each array unit performing part of the processing. The high I/O rates of an array unit make such operation efficient.

SECTION 4 - CONCLUSIONS

Item 2.10 of section 2 shows that about 4 seconds are required to process the SAR data from one synthetic aperture length with the configuration described in item 1.3.2 of section 1. This is about 5.7 times the real-time input rate (0.7 seconds are required to traverse one synthetic aperture length).

During this time the data for 1760 radar pulses are treated. An equivalent of 438 million butterfly operations are performed. The typical butterfly comprises six adds and four multiplications for a total of 4.38 billion real arithmetic operations. FFT arithmetic uses only about 1.4 seconds (35% of the 4 second processing time) so the average processing rate while executing FFT's is 3.12 billion real arithmetic operations per second. The moves required by the FFT's add 1.2 seconds per aperture (30% of the processing time) and the normalizations add about 6% to the processing time so the FFT processing rate drops to about one-half or 1.5 billion real arithmetic operations per second when these support operations are added.

Range processing is more efficient than azimuth processing - a reflection of the better layout for the range processing. For azimuth processing the layout of the array unit data has to be optimized for azimuth FFT operations, range curvature interpolations, and azimuth inverse FFT operations. The array unit memory is not large enough to allow a layout which is good for all operations.

Overall, data moves account for about half of the processing cycles. A larger array memory (more bits per PE) would reduce move times and also allow more scalar multiplications in the FFT's. Populating the array unit with 4096 bits per PE will reduce move time to about 20% of the execution time and halve the total execution time. An ARU with 4096 bits per PE will execute the SAR algorithm in 3 times real time.

A real-time configuration for the baseline mission is shown in Figure 10. Each array unit has 4096 bits of storage per PE. Staging memory 1 buffers and reformats the SAR input data for range processing by array unit 1. The range processed samples are collected in staging memory 2 and reformatted for azimuth processing by array unit 2 (or by array units 2 and 3). The fully processed samples are then collected in staging memory 3 where they are rearranged in a convenient format for the display.

The ADSPS is based on the MPP so all of the MPP software is available to implement SAR processing algorithms. System software

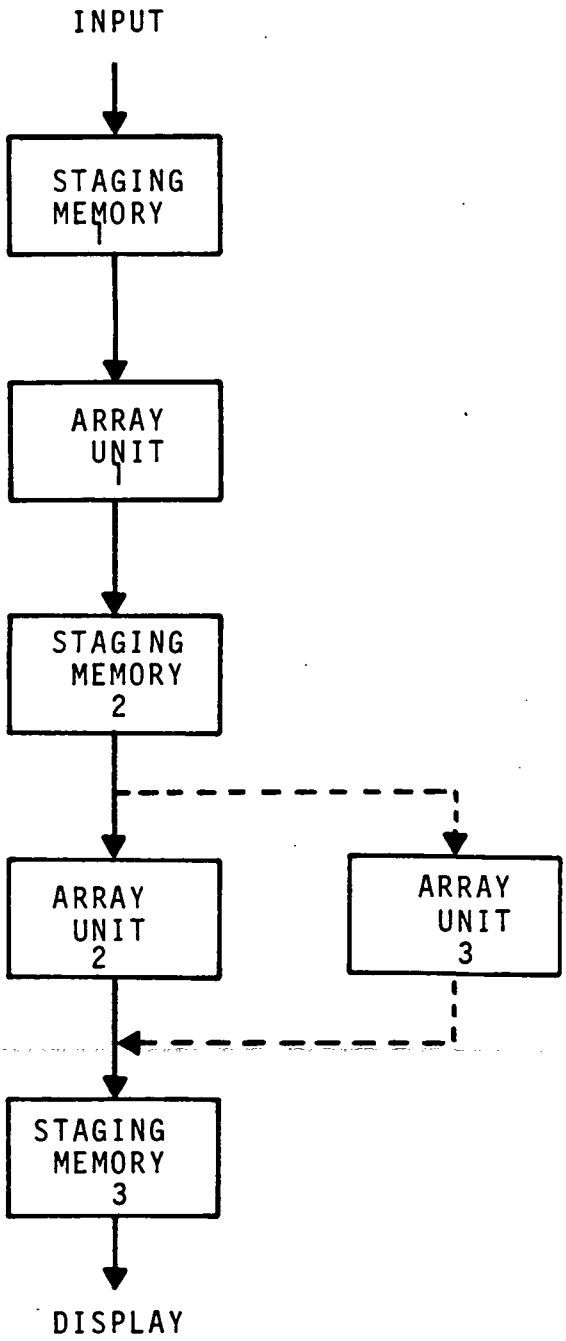

Figure 10 - Real Time SAR Processing System

in the MPP includes two macro-assemblers: the main control assembler to assemble application programs and the PE control assembler to add user written micro-routines to the set of system subroutines in the PE control unit memory. A linker and a librarian are included to facilitate the linking of separately assembled modules with modules from one or more libraries. The PDP-11 editor can be used to generate program source files. The control and debug module (CAD) loads programs into the MPP and facilitates their debugging. The staging memory manager manages the flow and reformatting of data through the staging memories.

To make the ADSPS a general-purpose SAR processor a number of modules should be added to the MPP system software. Each SAR source device should have a driver to govern the inputting of SAR data to the input staging memory. Similarly the display output device should have a driver to control the display of processed data. Since most SAR processing algorithms make extensive use of the fast Fourier transform a general-purpose FFT package is desirable. The package should handle a range of operand lengths and FFT sizes. The FFT size should range from 64 to 16384 and equal a power of two or a small odd multiple of a power of two. Other packages can be added to handle other common SAR operations such as range walk and curvature correction, reference function generation, complex-number magnitude determination, and the combination of multiple looks. With these packages the task of generating a SAR processing program for a particular mission is considerably simplified.

Appendix A

ANALYSIS OF SAR MISSION REQUIREMENTS

1. PURPOSE

The processing of synthetic aperture radar (SAR) signals is of appreciable interest to NASA Many NASA missions will employ SAR sensors. The variety of the SAR missions is illustrated in the data provided in Table I of the technical requirements. (For reader convenience this table is repeated as Table I in this section). Table I presents the SAR data in terms of mission requirements. In considering the digital SAR processing it is more useful to have the data expressed in terms appropriate to the data processing.

The analyses provided in the next item serve to develop processing related parameters from the mission related parameters of Table I. In this way the relative complexities of the different missions can be compared with respect to various aspects of the processing load. In particular data related to the Earth Resources SAR (ERSAR-L) mission and the SEASAT mission, which are similar to the baseline mission, can be compared with the other missions. It will be shown that these two missions represent by far the most pressing processing loads. Hence if the MPP can meet the baseline mission requirements, which encompass the ERSAR-L and SEASAT requirements, all other missions cán be accommodated.

2.0 ANALYSIS PROCEDURES

2.1 Approach

The first step in converting the mission parameters of Table I to processing related parameters is to compute the wavelengths corresponding to the RF frequencies. This computation is performed in order to permit further calculations to be made in terms of distance quantities. The wavelength is computed according to the equation:

$$\lambda = \frac{c}{f} \qquad (1)$$

where c is the velocity of light and has been taken as 3.0E10 centimeters per second.

Next the swath width in pixels, is determined by

dividing the swath width as given in kilometers by the resolution. This operation places the swath width in processor-related terms.

The slant range must now be determined from the given altitude and incident angle values. This computation can be carried out with the aid of the relations shown in Figure 1.

From this figure making use of the law of sines:

$$\cos \beta = \frac{R \sin \alpha}{R+h} \qquad (2)$$

Also,

$$\theta = \beta + \alpha - \pi/2$$

And thus,

$$R_s = \frac{R \sin \theta}{\cos \beta} \qquad (3)$$

In applying equation (3) a value of 6378 kilometers was used for the radius of the earth for the earth satellite missions, a value of 6050 kilometers was used for the radius of Venus, and a 4/3 earth radius value of 8504 kilometers was used for the two aircraft missions (aircraft L and aircraft X) to account for atmospheric refraction. In the satellite missions atmospheric refraction can be ignored since most of the radar propagation path is above the earth's atmosphere.

The maximum slant ranges for the two aircraft missions were taken as the horizon limit values since the minimum incident angle was given as 0 degrees. Two values of maximum slant range were then computed corresponding to the two altitude values given in Table I.. Although these slant range values still result in lower processing loads for the aircraft missions than the SEASAT and ERSAR-L missions, the horizon limit slant range may yield processing parameters that are more stringent than actually needed for the aircraft missions.

A major processing-related parameter, the synthetic aperture length (LS) may now be determined:

$$L_s = \frac{K \lambda R_s}{2W} \qquad (4)$$

K = Aperture Constant
W = Desired Resolution

The value of the aperture constant could theoretically be taken as unity. In practice a somewhat larger value is required for K to account for broadening of the main-lobe due to various phase errors and to permit shapening of the synthetic array's sidelobe characteristics. A somewhat typical value of 1.25 was used for K in the computations.

The azimuth compression ratio can be determined by dividing the synthetic aperture length by the required resolution. In the cases of the Aircraft X, VOIR Lo-Res, and ICEX missions it was found that the mission requirements in some cases could be met with single-look compression ratios less than unity. That is the desired azimuth resolution could be achieved without SAR processing in the normal sense - only the azimuth bandpass prefilter operations needed to separate the different required azimuth looks are necessary.

A quantity called the "total azimuth compression ratio" may now be found by multiplying the single-look compression ratio by the number of looks. In the case of the ERSAR missions the result is also multiplied by 2 to account for the dual polarization. The total azimuth compression ratio has no real physical significance but is an indication of the extent of the total azimuth processing required for the mission.

A second parameter of major significance to the SAR processing may now be computed with the aid of Figure 2. In Figure 2, the origin is taken at the SAR platform, motion is in the X direction, and the figure is drawn in the slant range plane. The slant range to any target point is given by:

$$R_s = \sqrt{x^2+y^2} = y\sqrt{1+\frac{x^2}{y^2}}$$

If $y \gg x$ (usual case) (5)

$$R_s = y + 1/2\,\frac{x^2}{y} = R_{so} + 1/2\,\frac{x^2}{R_{so}}$$

49

Now,

$$\theta = \text{radar phase angle} = 4\pi \frac{R_s}{\lambda}$$

$$= \frac{4\pi R_{so}}{\lambda} + \frac{2\pi x^2}{\lambda R_{so}}$$

$$\dot{\theta} = \frac{4\pi x v}{\lambda R_{so}}$$

$$f_a = \text{Doppler frequency} = \frac{1}{2\pi}\dot{\theta}$$

$$= \frac{2xv}{\lambda R_{so}}$$

The maximum f_a occurs for $x = \frac{Ls}{2}$

$$f_a \max = \frac{LsV}{\lambda R_{so}}$$

The PRF must be at least twice the maximum doppler frequency in order to eliminate azimuth ambiguities. Hence,

$$PRF = \frac{2LsV}{\lambda R_{so}} \qquad (6)$$

Substituting the value of Ls from equation (4)

$$PRF = \frac{KV}{W} \qquad (7)$$

The reason that the required PRF is a function of the required resolution is that as the required resolution W is reduced, the length of the phase history record which must be examined and the length of the synthetic aperture must be increased. These considerations lead to the examination of higher doppler frequencies with the attendant higher PRF.

The product of the PRF and the swath width in pixels provides the pixel per second processor input rate. Note that if D is the distance traveled during one PRF period then,

$$D = \frac{V}{PRF}$$

Substituting from equation (7)

$$D = \frac{W}{K} \qquad (8)$$

Thus since K is greater than unity, the distance traveled during a PRF period is less than the required resolution. The input pixel rate is also higher than the product of swath width in pixels by the platform velocity in pixels per second (an input pixel rate which casual observation might indicate) by the factor K. As previously indicated, K must be larger than unity to account for phase errors and to provide azimuth sidelobe control. However after account has been made for these considerations in the earlier stages of the processing, the balance of the processing could proceed at a pixel spacing of W. However the displayed output image should be generated at a pixel spacing less than W to yield a true resolution of W in the image. This result follows because of granularity effects in the digitally produced image.

It should be noted that because of peak transmitter power capacity or other considerations, the actual transmitted PRF may be higher than that indicated by equation (6) and (7). In such cases the actual PRF is immediately reduced to the minimum value as given by equations (6) and (7) at the start of the digital SAR processing by the azimuth prefilter operations. In this way processing requirements for the balance of the operations are kept at minimum values.

The value of V used in equations (6) and (7) was taken to be the orbital velocity of a satellite in circular orbit in the case of the satellite missions and was taken to be 300 meters per second in the case of the Aircraft L mission. The orbital velocity of a satellite in circular orbit is given by:

$$V = \sqrt{\frac{u}{r}}$$

where u = Gravitational constant
(3.986E5 Km^3/sec^2 for earth orbits)
r = Radius of circular orbit

Inserting the value of u for the earth,

$$V = \frac{631.35}{\sqrt{r}} \text{ Km/sec (earth orbit)} \qquad (9)$$

Values for the quantities discussed above are provided in Table II for the different mission sets. The "Total Input Pixels per second" was obtained by multiplying the "Input Pixels per second

single-look" by the number of looks. The Total Input Pixels per Second is thus indicative of the processing rate requirement. However the processing in the azimuth direction is also related to the azimuth compression ratio. The processing algorithm (to be determined later) will probably make use of some sort of FFT processing. This FFT processing per pixel is proportional to the base 2 logarithm of the azimuth compression ratio per look. The final entries in Table II provide a processing index computed by multiplying the pixels per second by the base 2 logarithm of the azimuth compression ratio. It is felt that this processing index is indicative of the relative processing load of each mission.

In using the processing index described above it is assumed that the extent of the range compression processing is the same for each mission. This assumption is probably not quite correct. Since no data was provided for the range compression (except for the baseline mission) it is not possible to make relative computations relative to the range compression. Since the range compression computations are independent of the total range, these computations may be assumed to represent, to a first approximation, an equal processing load for all missions having nearly equal resolutions. Hence the processing index in Table II can be taken as a valid indicator of the relative processing load of the different missions.

2.2 Conclusions

From the Processing Index given in Table II, it is seen that the processing loads for both single and multiple-look images represented by the SEASAT and ERSAR L missions are similar and appreciably higher than the processing loads of the other missions. The processing load of the ERSAR L mission is about 28 percent higher than that of SEASAT because of the higher resolution requirements of the ERSAR L mission.

The baseline mission parameters, as presented in the technical requirements, are similar to those of ERSAR L although the swath width is closer to the SEASAT mission. Consequently a processing system which meets the baseline mission requirements should have the capability to meet the other mission requirements readily.

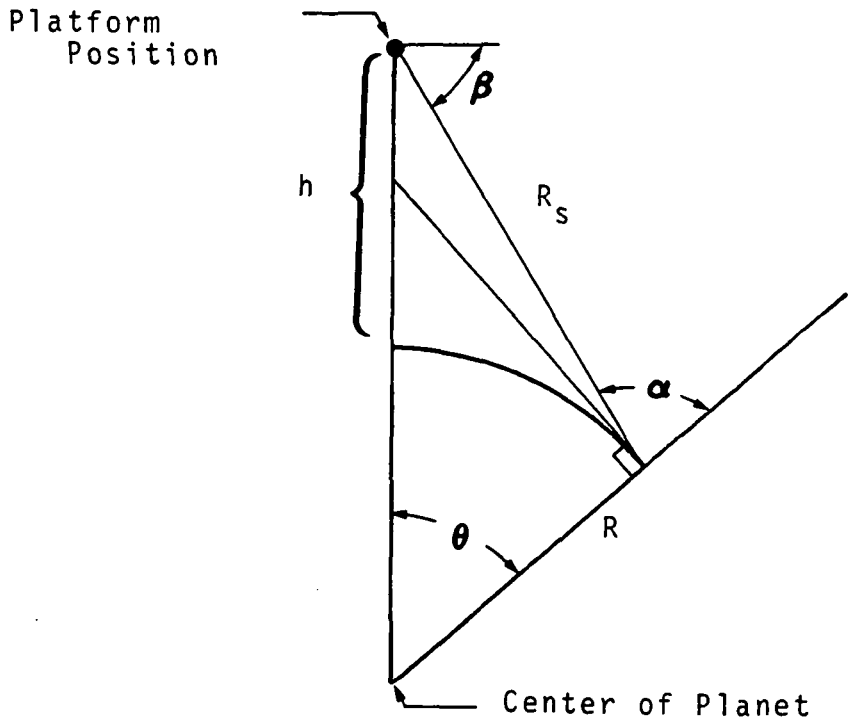

Figure 1 - Geometry for Slant Range Determination

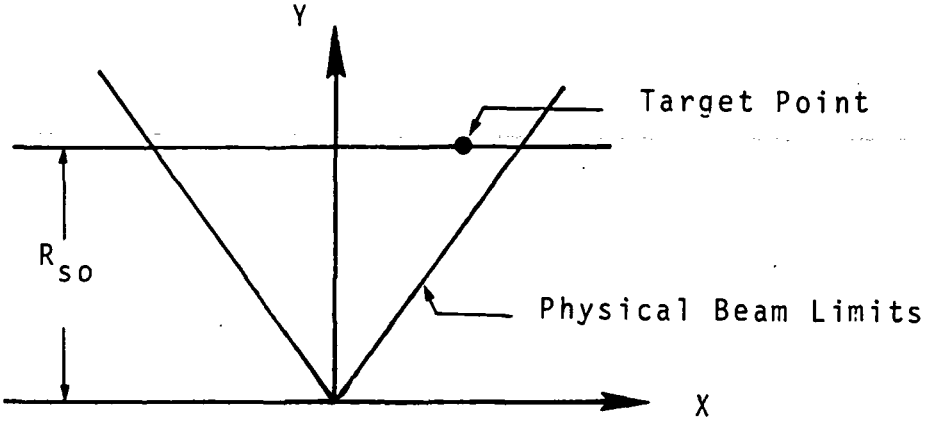

Figure 2 - Relations for Determining Doppler Frequencies

TABLE I - SAR MISSION SET

Mission Set	Resolution m	Swath Width km	Number of Looks	Incident Angle Degrees	Frequency MHz	Polarization*	Altitude km
SEASAT	25	100	4	22.54	1275	S	800
SIR REFLIGHT	40	50	7	20-70	1275	S	210-280
AIRCRAFT L	10-20	18-35	8	0-60	1275	S	3-12
X	10-20	18-35	8	0-60	9600	S	3-12
VOIR HI RES 52°	75	TBD	4	52	1275	S	250
LO RES 52°	300	30	30	52	1275	S	250
ICEX	150	360	6	36.42	9600	S	700
**ERSAR L	15	40	4	60°	1275	D	300
**ERSAR X	15	40	4	60°	9600	D	300

*S = Single, D = Dual

**Preliminary

GOODYEAR AEROSPACE

TABLE II - SAR MISSION DATA PROCESSING PARAMETERS

Mission set	Wavelength (cm)	Swath width (pixels)	Max Slant range (km)	Synthetic aperture lenght (meters)	Azimuth compression ratio per look	Total azimuth compression ratio	PRF per look (Hz)	Input pixels per second, Single Look (millions)	Total Input pixels per second, (millions)	Processing Index	
										Single Look	Total
SEASAT	23.5	4000	860	5040	200	810	373	1.5	6.0	11.4	45.6
SIR reflight	23.5	1250	720	2640	65	460	242	0.3	2.1	1.83	12.8
Aircraft L	23.5	900 to 3500	3 to 24	22 to 352	1.1 to 35.2	8.9 to 281	45	0.04 to 0.16	0.32 to 1.3	0.81	6.5
Aircraft X	3.13	900 to 3500	3 to 24	2.9 to 46.9	0.15 to 4.69	1.2 to 37.5	*	*	*	*	*
VOIR hi-res	23.5	-	390	770	10	40	*	*	*	*	*
VOIR lo-res	23.5	100	390	190	0.64	19.3	*	*	*	*	*
ICEX	3.13	2400	850	110	0.74	4.4	*	*	*	*	*
ERSAR L	23.5	2667	560	5520	370	1470	644	1.7	6.9	14.6	58.5
ERSAR X	3.13	2667	560	740	49	200	*	*	*	*	*

*These values were not computed because these missions appear to present the fewest demands on the processor.

APPENDIX B

Relatively Prime FFT (Review)

The Discrete Fourier Transform (DFT) is described by

1) $y_m = \frac{1}{N} \sum_{k=0}^{N-1} X_k W_N^{km}$ where $W_N^{km} = e^{-i2\pi km/N}$ and

$k, m = 0, ----, (N-1)$.

If N can be written as the product of two integers (N_1 and N_2) which are relatively prime, then any unique ℓ in the range, $0 \leq \ell \leq N$, can be written in the form

2a) $\ell = (\alpha_1 N_1 + \alpha_2 N_2)_{\text{MOD } N}$ where α_1 and α_2 are integers and

$$\left\{ \begin{array}{l} 0 \leq \alpha_1 \leq (N_2 - 1) \\ 0 \leq \alpha_2 \leq (N_1 - 1) \end{array} \right\}$$

In like manner, m can be written as

2b) $m = \beta_1 N_1 + \beta_2 N_2)$ where β_1 and β_2 are integers and

$$\left\{ \begin{array}{l} 0 \leq \beta_1 \leq (N_2 - 1) \\ 0 \leq \beta_2 \leq (N_1 - 1) \end{array} \right\} .$$

From 2a) and 2b),

3a) $\ell + \gamma_1 N = \alpha_1 N_1 + \alpha_2 N_2$

3b) $m + \gamma_2 N = \beta_1 N_1 + \beta_2 N_2$, γ_1, γ_2 are integers, and so

4a) $W_N^{(\ell + \gamma_1 N)(m + \gamma_2 N)} = W_N^{\ell m} = W_N^{(\alpha_1 N_1 + \alpha_2 N_2)(\beta_1 N_1 + \beta_2 N_2)}$

4b) $= W_N^{(\alpha_1 \beta_1 N_1^2 + \alpha_2 \beta_2 N_2^2)}$

4c) $= W_{N_2}^{\alpha_1 (\beta_1 N_1)} * W_{N_1}^{\alpha_2 (\beta_2 N_2)}$

using 4c) in 1),

$$5)\quad y_{m(\beta_1,\beta_2)} = \left(\frac{1}{N_1} \sum_{\alpha_2=0}^{N_1-1} \left(\frac{1}{N_2} \sum_{\alpha_1=0}^{N_2-1} x_{\ell(\alpha_1,\alpha_2)} W_{N_2}^{\alpha_1 \gamma_1} \right) W_{N_1}^{\alpha_2 \gamma_2} \right)$$

where $\begin{cases} \gamma_1 = (\beta_1 N_1)_{MOD\ N_2} & \text{or } \beta_1 N_1 = \gamma_1 + \mu_1 N_2 \\ \gamma_2 = (\beta_2 N_2)_{MOD\ N_1} & \text{or } \beta_2 N_2 = \gamma_2 + \mu_2 N_1, \end{cases}$

μ_1 and μ_2 are integers.

The inner bracketed term of 5) describes N_1 FFT's of size N_2 (one for each value of α_2). The outer bracketed term describes N_2 FFT's of size N_1 (one for each value of γ_1(or β_1)).

Note that the FFT's of size N_1 could be performed first since 5) can be written as

$$6)\quad y_{m(\beta_1,\beta_2)} = \left(\frac{1}{N_2} \sum_{\alpha_1=0}^{N_2-1} \left(\frac{1}{N_1} \sum_{\alpha_2=0}^{N_1-1} x_{\ell(\alpha_1,\alpha_2)} W_{N_1}^{\alpha_2 \gamma_2} \right) W_{N_2}^{\alpha_1 \gamma_1} \right)$$

No twiddle factors are required because N_1 and N_2 are relatively prime.

Consider, now, the specific SAR example of an N=5120 point transform. Since N can be written as $N=N_1*N_2$ (where $N_1=5$ and N_2 = 1Ø24) and N_1, N_2 are relatively prime, equations 5) or 6) can be used to accomplish the transform. Arbitrarily, 5) will be used first as the basis for executing the DFT.

Using 2a),

$$\ell = (5\,\alpha_1 + 1024\,\alpha_2) \text{ where}$$

$$\phi \leq \alpha_1 < 1024$$

$$\phi \leq \alpha_2 < 5 \quad .$$

Table IA shows ℓ vs. α_1 for fixed α_2; Table IB shows ℓ vs. α_2 for fixed α_1.

Table IA - The ℓ-index vs. α_1 (Fixed α_2)

	$\alpha_2=0$	$\alpha_2=1$	$\alpha_2=2$	$\alpha_2=3$	$\alpha_2=4$
α_1	ℓ	ℓ	ℓ	ℓ	ℓ
0	0	1024	2048	3072	4096
1	5	1029	2053	3077	5001
2	10	1034	2058	3082	5006
3	15	1039	2063	3087	5011
4	20	1044	2068	3092	5016
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
1023	5115	1019	2043	3067	4091

Table IB - The ℓ-index vs. α_2 (Fixed α_1)

α_2	$\alpha_1=0$	$\alpha_1=1$	$\alpha_1=2$	$\alpha_1=205$	$\alpha_1=410$	$\alpha_1=615$	$\alpha_1=820$	$\alpha_1=1023$
	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ
0	0	5	10	1025	2050	3075	4100 (5124)	5115
1	1024	1029	1034	2049	3074	4099 (5123)	4 (6148)	1019
2	2048	2053	2058	3073	4098 (5122)	3 (6147)	1028 (7172)	2043
3	3072	3077	3082	4097 (5121)	2 (6146)	1027 (7171)	2052 (8196)	3067
4	4096	4101	4106	1	1026	2051	3076	4091

Since α_1 can assume 1024 different values, 1024 inner 5-point DFT's are performed to produce the inner bracketed value of 6). During any one of the inner DFT's, α_1 is held constant. By using 2a), it is possible to identify the input sample index, ℓ, for each α_2 value. These indices are shown in Table IB and are used to locate the corresponding complex sample values that are stored in the MPP memory.

Appendix C - The DFT* of Small Dimension Vectors

Generally, a DFT of a high dimension vector can be described as a succession of DFT's of vectors of smaller dimension.

Explicity, the DFT of X, a large dimensioned N component vector, yields a like dimensioned vector, Y, whose elements, y_k, are given by

$$(C1) \qquad y_k = \sum_{\ell=0}^{N-1} x_\ell w_N^{k\ell} \qquad \text{where} \quad w_N^{k\ell} = e^{-j(2\pi k\ell/N)}$$

Suppose $N=L*K$. Then if k is described by $k=k_0+k_1K$ and $\ell=\ell_0+\ell_1 L$, the elements of Y are given by

$$(C2) \qquad y_{k_0+k_1K} = \sum_{\ell_0=0}^{L-1} \left(w_N^{k_0\ell_0} \left(\sum_{\ell_1=0}^{K-1} x_{\ell 0+\ell_1 L} \, w_K^{\ell_1 K_0} \right) \right) w_L^{\ell_0 k_1}$$

The inner bracketed term simply defines a DFT of dimension K for each ℓ_0 (L such values) whereas the outer summation defines a DFT of dimension L for each k_0 (K such values). By performing the L DFT's of dimension K and then performing the K DFT's of dimension L on the results, Y is obtained. Thus a large dimensioned DFT can be accomplished by executing a sequence of short DFT's. This appendix shows how various short DFT's can be realized.

* DFT = Discrete Fourier Transform

The DFT of a vector of dimension N is simply a linear transformation of the vector X into a new space. The expression (C1) in matrix form is given by

(C3) Y=WX where the k-indexed row, ℓ-indexed column element of W is $w_N^{k\ell}$. The exponent array of W, namely, S, is defined to be the array whose elements are the exponents of the elements of W. Let LLOG be the operator that strips off the exponents. Then

$$S=LLOG(W)= [LLOG(W_N^{k\ell})] = [k\ell]$$
$$N*N \qquad N*N$$

which is displayed in Figure C1. When dealing with the elements of W, it makes no difference whether $k*\ell$ is treated as a Modulo N number or a Modulo $(N**2)$ number. Let the operator MOD(S,N) imply that all elements of S are reduced to modulo N numbers.

FIGURE C2 displays B=MOD(S,N) for various small N's.To emphasize conjugate symmetry properties, several of the smaller B_N's are defined using negative modulo N numbers.

The inverse operator, INVLLOG, acting on B_N, restores the W matrix with dimension N. Let the inverted carrot symbol, $\wedge$, represent this operator. Then $W= \hat{B}_N$.
When restoring W using the FIGURE (B2) displays, the symmetry properties and values of Table C1 are useful.

The re-inverted B_N's, namely $\widehat{B}_N$'s, are shown in Figure C3.

FIGURE C1 - Expanded form of LLOG(W)

$$S=\begin{bmatrix} 0*0 & 0*1 & 0*2 & 0*3 & 0*4 & \cdots & 0*(N-1) \\ 1*0 & 1*1 & 1*2 & 1*3 & 1*4 & \cdots & 1*(N-1) \\ 2*0 & 2*1 & 2*2 & 2*3 & 2*4 & \cdots & 2*(N-1) \\ 3*0 & 3*1 & 3*2 & 3*3 & 3*4 & \cdots & 3*(N-1) \\ 4*0 & 4*1 & 4*2 & 4*3 & 4*4 & \cdots & 4*(N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ N-1)*0 & (n-1)*2 & (N-1)*2 & (N-1)*3 & (N-1)*4 & \cdots & (N-1)*(N-1) \end{bmatrix}_{N*N}$$

In more compact form,

$$S=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & \cdots & (N-1) \\ 0 & 2 & 4 & 6 & 8 & 10 & \cdots & 1*(N-1) \\ 0 & 3 & 6 & 9 & 12 & 15 & \cdots & 3*(N-1) \\ 0 & 4 & 8 & 12 & 16 & 20 & \cdots & 4*(N-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & (N-1) & 2*(N-1) & 3*(N-1) & 4*(N-1) & & & (N-1)*(N-1) \end{bmatrix}_{N*N}$$

FIGURE C2 - MOD(S,N) or N=3,---,16

$$B_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}_{3*3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}_{3*3}$$

$$B_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 0 & 2 \\ 0 & 3 & 2 & 1 \end{bmatrix}_{4*4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & -1 \\ 0 & 2 & 0 & 2 \\ 0 & -1 & 2 & 1 \end{bmatrix}_{4*4}$$

$$B_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 1 & 3 \\ 0 & 3 & 1 & 4 & 2 \\ 0 & 4 & 3 & 2 & 1 \end{bmatrix}_{5*5} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & -2 & -1 \\ 0 & 2 & -1 & 1 & -2 \\ 0 & -2 & 1 & -1 & 2 \\ 0 & -1 & -2 & 2 & 1 \end{bmatrix}_{5*5}$$

$$B_6 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 4 & 0 & 2 & 4 \\ 0 & 3 & 0 & 3 & 0 & 3 \\ 0 & 4 & 2 & 0 & 4 & 2 \\ 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}_{6*6} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & -2 & -1 \\ 0 & 2 & -2 & 0 & 2 & -2 \\ 0 & 3 & 0 & 3 & 0 & 3 \\ 0 & -2 & 2 & 0 & -2 & 2 \\ 0 & -1 & -2 & 3 & 2 & 1 \end{bmatrix}_{6*6}$$

$$B_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & -3 & -2 & -1 \\ 0 & 2 & -3 & -1 & 1 & 3 & -2 \\ 0 & 3 & -1 & 2 & -2 & 1 & -3 \\ 0 & -3 & 1 & -2 & 2 & -1 & 3 \\ 0 & -2 & 3 & 1 & -1 & -3 & 2 \\ 0 & -1 & -2 & -3 & 3 & 2 & 1 \end{bmatrix}_{7*7}$$

$$B_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & -3 & -2 & -1 \\ 0 & 2 & 4 & -2 & 0 & 2 & 4 & -2 \\ 0 & 3 & -2 & 1 & 4 & -1 & 2 & -3 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & 4 \\ 0 & -3 & 2 & -1 & 4 & 1 & -2 & 3 \\ 0 & -2 & 4 & 2 & 0 & -2 & 4 & 2 \\ 0 & -1 & -2 & -3 & 4 & 3 & 2 & 1 \end{bmatrix}_{8*8}$$

FIGURE C2 - (continued)

$B_9 =$

0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8
0	2	4	6	8	1	3	5	7
0	3	6	0	3	6	0	3	6
0	4	8	3	7	2	6	1	5
0	5	1	6	2	7	3	8	4
0	6	3	0	6	3	0	6	3
0	7	5	3	1	8	6	4	2
0	8	7	6	5	4	3	2	1

9*9

$B_{10} =$

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	0	2	4	6	8
0	3	6	9	2	5	8	1	4	7
0	4	8	2	6	0	4	8	2	6
0	5	0	5	0	5	0	5	0	5
0	6	2	8	4	0	6	2	8	4
0	7	4	1	8	5	2	9	6	3
0	8	6	4	2	0	8	6	4	2
0	9	8	7	6	5	4	3	2	1

10*10

$B_{11} =$

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10
0	2	4	6	8	10	1	3	5	7	9
0	3	6	9	1	4	7	10	2	5	8
0	4	8	1	5	9	2	6	10	3	7
0	5	10	4	9	3	8	2	7	1	6
0	6	1	7	2	8	3	9	4	10	5
0	7	3	10	6	2	9	5	1	8	4
0	8	5	2	10	7	4	1	9	6	3
0	9	7	5	3	1	10	8	6	4	2
0	10	9	8	7	6	5	4	3	2	1

11*11

$B_{12} =$

0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11
0	2	4	6	8	10	0	2	4	6	8	10
0	3	6	9	0	3	6	9	0	3	6	9
0	4	8	0	4	8	0	4	8	0	4	8
0	5	10	3	8	1	6	11	4	9	2	7
0	6	0	6	0	6	0	6	0	6	0	6
0	7	2	9	4	11	6	1	8	3	10	5
0	8	4	0	8	4	0	8	4	0	8	4
0	9	6	3	0	9	6	3	0	9	6	3
0	10	8	6	4	2	0	10	8	6	4	2
0	11	10	9	8	7	6	5	4	3	2	1

12*12

FIGURE C2 - (continued) Sheet 3 of 3)

B_{13} =

0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12
0	2	4	6	8	10	12	1	3	5	7	9	11
0	3	6	9	12	2	5	8	11	1	4	7	10
0	4	8	12	3	7	11	2	6	10	1	5	9
0	5	10	2	7	12	4	9	1	6	11	3	8
0	6	12	5	11	4	10	3	9	2	8	1	7
0	7	1	8	2	9	3	10	4	11	5	12	6
0	8	3	11	6	1	9	4	12	7	2	10	5
0	9	5	1	10	6	2	11	7	3	12	8	4
0	10	7	4	1	11	8	5	2	12	9	6	3
0	11	9	7	5	3	1	12	10	8	6	4	2
0	12	11	10	9	8	7	6	5	4	3	2	1

13*13

B_{14} =

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	2	4	6	8	10	12	0	2	4	6	8	10	12
0	3	6	9	12	1	4	7	10	13	2	5	8	11
0	4	8	12	2	6	10	0	4	8	12	2	6	10
0	5	10	1	6	11	2	7	12	3	8	13	4	9
0	6	12	4	10	2	8	0	6	12	4	10	2	8
0	7	0	7	0	7	0	7	0	7	0	7	0	7
0	8	2	10	4	12	6	0	8	2	10	4	12	6
0	9	4	13	8	3	12	7	2	11	6	1	10	5
0	10	6	2	12	8	4	0	10	6	2	12	8	4
0	11	8	5	2	12	10	7	4	1	12	9	6	3
0	12	10	8	6	4	2	0	12	10	8	6	4	2
0	13	12	11	10	9	8	7	6	5	4	3	2	1

14*14

B_{16} =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	14	0	2	4	6	8	10	12	14
0	3	6	9	12	15	2	5	8	11	14	1	4	7	10	13
0	4	8	12	0	4	8	12	0	4	8	12	0	4	8	12
0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11
0	6	12	2	8	14	4	10	0	6	12	2	8	14	4	10
0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9
0	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8
0	9	2	11	4	13	6	15	8	1	10	3	12	5	14	7
0	10	4	14	8	2	12	6	0	10	4	14	8	2	12	6
0	11	6	1	12	7	2	13	8	3	14	9	4	15	10	5
0	12	8	4	0	12	8	4	0	12	8	4	0	12	8	4
0	13	10	7	4	1	14	11	8	5	2	15	12	9	6	3
0	14	12	10	8	6	4	2	0	14	12	10	8	6	4	2
0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

16*16

B_{15} =

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	2	4	6	8	10	12	14	1	3	5	7	9	11	13	
0	3	6	9	12	0	3	6	9	12	0	3	6	9	12	
0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	
0	5	10	0	5	10	0	5	10	0	5	10	0	5	10	
0	6	12	3	9	0	6	12	3	9	0	6	12	3	9	
0	7	14	6	13	5	12	4	11	3	10	2	9	1	8	
0	8	1	9	2	10	3	11	4	12	5	13	6	14	7	
0	9	3	12	6	0	9	3	12	6	0	9	3	12	6	
0	10	5	0	10	5	0	10	5	0	10	5	0	10	5	
0	11	7	3	14	10	6	2	13	9	5	1	12	8	4	
0	12	9	6	3	0	12	9	6	3	0	12	9	6	3	
0	13	11	9	7	5	3	1	14	12	10	8	6	4	2	
0	14	13	12	11	10	9	8	7	6	5	4	3	2	1	

15*15

Table C1 - Values of W Elements Commonly Used

$W^N = (1,0)$

$W^{N/2} = -W^N = (-1,0)$

$W^{N/3} = (-a,-b)$
$W^{2N/3} = (W^{N/3})^* = (-a,b)$

where $a = \cos 60^\circ = \sin 30^\circ$ $\quad b = \sin 60^\circ = \cos 30^\circ$

$W^{N/4} = -i\,(W^N) = (0,-1)$
$W^{3N/4} = (W^{N/4})^* = 0,1)$

$W^{N/5} = (c,-d)$
$W^{2N/5} = (-e,-f)$
$W^{3N/5} = (W^{2N/5})^* = (-e,f)$
$W^{4N/5} = (W^{N/5})^* = (c,d)$

where $c = \cos 72^\circ = \sin 18^\circ$, $\quad d = \sin 72^\circ = \cos 18^\circ$
$e = \cos 36^\circ$, $\quad f = \sin 36^\circ$

Note: $\cos 36 - \cos 72 = 1/2 = 2 \cos 36 \cos 72$

$\cos 72 = \sin 18 = (\sqrt{5}-1)/4$

$\cos 36 = (\sqrt{5}+1)/4$

$W^{N/6} = (a,-b)$
$W^{5N/6} = (W^{N/6}) = (a,b)$

$W^{N/7} = (g,-h)$
$W^{2N/7} = (-o,-p)$
$W^{3N/7} = (-q,-r)$
$W^{4N/7} = (W^{3N/7})^* = (-g,r)$
$W^{5N/7} = (W^{2N/7})^* = (-o,p)$
$W^{6N/7} = (W^{N/7})^* = (g,h)$

where $g = \cos(51\ 3/7^\circ) = \sin(38\ 4/7^\circ)$ $\quad h = \sin(51\ 3/7^\circ) = \cos(38\ 4/7^\circ)$
$o = \cos(77\ 1/7^\circ) = \sin(12\ 6/7^\circ)$ $\quad p = \sin(77\ 1/7^\circ) = \cos(12\ 6/7^\circ)$
$q = \cos(25\ 5/7^\circ)$ $\quad r = (\sin 25\ 5/7^\circ)$

$W^{N/8} = (s,-t)$
$W^{3N/8} = (-i\ W^{N/8}) = (-t,-s)$
$W^{5N/8} = (W^{3N/8})^* = (-t,s)$
$W^{7N/8} = (s,t)$

where $s = \cos(45^\circ)$ $\quad t = \sin(45^\circ)$

Table C1 (Continued)

$W^{N/9} = (u,-v)$

$W^{2N/9} = (w,-x)$

$W^{4N/9} = (-y,-z)$

$W^{5N/9} = (W^{4N/9})^* = (-y,z)$

$W^{7N/9} = (W^{2N/9})^* = (w,x)$

$W^{8N/9} = (W^{N/9})^* = (u,v)$

where $u=\cos 40^\circ$, $v=\sin 40^\circ$

$w=\cos 80^\circ$, $x=\sin 80^\circ$

$=\sin 10^\circ$ $\quad \cos 10^\circ$

$y=\cos 20^\circ$, $\sin 20^\circ$

$W^{N/10} = (e,-f)$

$W^{3N/10} = (-c,-d)$

$W^{7N/10} = (-c,d)$

$W^{9N/10} = (e,f)$

$W^{N/11} = (aa,-bb)$

$W^{2N/11} = (cc,-dd)$

$W^{3N/11} = (-ee,-ff)$

$W^{4N/11} = (-gg,-hh)$

$W^{5N/11} = (-oo,pp)$

$W^{6N/11} = (W^{5N/11})^*$

$W^{7N/11} = (W^{4N/11})^*$

$W^{8N/11} = (W^{3N/11})^*$

$W^{9N/11} = (W^{2N/11})^*$

$W^{10N/11} = (W^{N/11})^*$

where

aa = cos 32 8/11, bb=sin(32 8/11)

cc = cos 65 5/11, dd=sin(65 5/11)

ee = cos(81 9/11),ff=sin(81 9/11)

gg = cos(49 1/11),hh=sin(49 1/11)

oo = cos(16 4/11),pp=sin(16 4/11)

or

cc = sin(24 6/11),dd=cos(24 6/11)

ee = sin (8 2/11),ff=cos (8 2/11)

gg = sin(40 10/11)hh=cos(40 10/11)

Table C1 - (Continued)

$W^{N/12} = (b,-a)$

$W^{6N/12} = (-b,-a)$

$W^{7N/12} = (-b,a)$

$W^{11N/12} = (b,a)$

$W^{N/13} = (qq,-rr)$

$W^{2N/13} = (ss,-tt)$

$W^{3N/13} = (+uu,-vv)$

$W^{4N/13} = (-ww,-xx)$

$W^{5N/13} = (-yy,-zz)$

$W^{6N/13} = (-aaa,-bbb)$

$W^{7N/13} = (-aaa,bbb)$

$W^{8N/13} = (-yy,zz)$

$W^{9N/13} = (-ww,xx)$

$W^{10N/13} = (uu,vv)$

$W^{11N/13} = (ss,tt)$

$W^{12N/13} = (qq,rr)$

$W^{N/14} = (q,-r)$

$W^{3N/14} = (o,-p)$

$W^{5N/14} = (-g,-h)$

$W^{9N/14} = (-g,h)$

$W^{11N/14} = (o,p)$

$W^{13N/14} = (q,r)$

where
- qq = cos(27 9/13),rr=sin(27 9/13)
- ss = cos(54 5/13),tt=sin(54 5/13)
- uu = cos(53 1/13),vv=sin(83 1/13)
- ww = cos(69 3/13),xx=sin(69 3/13)
- yy = cos(41 7/13),zz=sin(41 7/13)
- aaa=cos(13 11/13),bbb=sin(13 11/13)

or

- ss=sin(35 8/13), tt=cos(35 8/13)
- uu=sin(6 12/13), vv=cos (6 12/13)
- ww=sin(20 10/13),xx=cos(20 10/13)

Table C1 - (Continued)

$W^{N/15} = (ccc, -ddd)$

$W^{2N/15} = (eee, -fff)$

$W^{4N/15} = (-ggg, -hhh)$

$W^{7N/15} = (-ooo, -ppp)$

$W^{8N/15} = (-ooo, ppp)$

$W^{11N/15} = (-ggg, hhh)$

$W^{13N/15} = (eee, fff)$

$W^{14N/15} = (ccc, ddd)$

where
- ccc=cos 24, ddd=sin 24
- eee-cos 48, fff=sin 48
- ggg=cos 84, hhh=sin 84
- ooo=cos 12, ppp=sin 12

or

- eee=sin 42, fff=cos 42
- ggg=sin 6, hhh=cos 6

$W^{N/16} = (\alpha, -\beta)$

$W^{3N/16} = (\beta, -\alpha)$

$W^{5N/16} = (-\beta, -\alpha)$

$W^{7N/16} = (-\alpha, -\beta)$

$W^{9N/16} = (-\alpha, \beta)$

$W^{11N/16} = (-\beta, \alpha)$

$W^{13N/16} = (\beta, \alpha)$

$W^{15N/16} = (\alpha, \beta)$

where $\alpha = \cos(22\ 1/2)$, $\beta = \sin(22\ 1/2)$

Figure C3 - Inverse B_N's (B_NS) for N = 3,....,16

$$= \hat{B}_3 = \begin{bmatrix} (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\ (1,\emptyset) & (-a,-b) & (-a,+b) \\ (1,\emptyset) & (-a,b) & (-a,-b) \end{bmatrix}$$

$$\hat{B}_4 = \begin{bmatrix} (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\ (1,\emptyset) & (\emptyset,-1) & (-1,\emptyset) & (\emptyset,1) \\ (1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) \\ (1,\emptyset) & (\emptyset,1) & (-1,\emptyset) & (\emptyset,-1) \end{bmatrix}$$

$$\hat{B}_5 \begin{bmatrix} (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\ (1,\emptyset) & (c,-d) & (-e,-f) & (-e,f) & (c,d) \\ (1,\emptyset) & (-e,-f) & (c,d) & (c,-d) & (-e,f) \\ (1,\emptyset) & (-e,f) & (c,-d) & (c,d) & (-e,-f) \\ (1,\emptyset) & (e,d) & (-e,f) & (-e,-f) & (c,-d) \end{bmatrix}$$

or

$$\hat{B}_6 = \begin{bmatrix} (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\ (1,\emptyset) & (-a,-b) & (-a,-b) & (-1,\emptyset) & (-a,b) & (a,b) \\ (1,\emptyset) & (-a,-b) & (-a,b) & (1,\emptyset) & (-a,-b) & (-a,b) \\ (1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) \\ (1,\emptyset) & (-a,b) & (-a,-b) & (1,\emptyset) & (-a,b) & (-a,-b) \\ (1,\emptyset) & (a,b) & (-a,b) & (-1,\emptyset) & (-a,-b) & (a,-b) \end{bmatrix}$$

$$\hat{B}_7 \begin{bmatrix} (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\ (1,\emptyset) & (g,-h) & (-o-p) & (-q,-r) & (-q,r) & (-o,p) & (g,h) \\ (1,\emptyset) & (-o,-p) & (-q,r) & (g,h) & (g,-h) & (-q,-r) & (-o,p) \\ (1,\emptyset) & (-q,-r) & (g,h) & (-o,-p) & (-o,p) & (g,-h) & (-q,r) \\ (1,\emptyset) & (-q,r) & (g,-h) & (-o,p) & (-o,-p) & (g,h) & (-q,-r) \\ (1,\emptyset) & (-o,p) & (-q,-r) & (g,-h) & (g,h) & (-q,r) & (-o,-p) \\ (1,\emptyset) & (g,h) & (-o,p) & (-q,r) & (-q,-r) & (-o,-p) & (-g,-h) \end{bmatrix}$$

Figure C3 - (Continued)

$$\hat{B}_8 = \begin{bmatrix}
(1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) & (1,\emptyset) \\
(1,\emptyset) & (s,-t) & (\emptyset,-1) & (-t,-s) & (-1,\emptyset) & (-t,s) & (\emptyset,1) & (s,t) \\
(1,\emptyset) & (\emptyset,-1) & (-1,\emptyset) & (\emptyset,1) & (1,\emptyset) & (\emptyset,-1) & (-1,\emptyset) & (\emptyset,1) \\
(1,\emptyset) & (-t,-s) & (\emptyset,1) & (s,-t) & (-1,\emptyset) & (s,t) & (\emptyset,-1) & (-t,s) \\
(1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) & (1,\emptyset) & (-1,\emptyset) \\
(1,\emptyset) & (-t,s) & (\emptyset,-1) & (s,t) & (-1,\emptyset) & (s,-t) & (\emptyset,1) & (-t,-s) \\
(1,\emptyset) & (\emptyset,1) & (-1,\emptyset) & (\emptyset,-1) & (1,\emptyset) & (\emptyset,1) & (-1,\emptyset) & (\emptyset,-1) \\
(1,\emptyset) & (s,t) & (\emptyset,1) & (t,s) & (-1,\emptyset) & (-t,-s) & (\emptyset,-1) & (s,-t)
\end{bmatrix}$$

(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)
(1,0)	$(\alpha,-\beta)$	(s,-s)	$(\beta,-\alpha)$	(0,-1)	$(-\beta,-\alpha)$	(-s,-s)	$(-\alpha,-\beta)$	(-1,0)	$(-\alpha,\beta)$	(-s,s)	$(-\beta,\alpha)$	(0,1)	(β,α)	(α,β)
(1,0)	(s,-s)	(0,-1)	(-s,-s)	(-1,0)	(-s,s)	(0,1)	(s,s)	(1,0)	(s,-s)	(0,-1)	(-s,-s)	(-1,0)	(-s,s)	(0,1)
(1,0)	$(\beta,-\alpha)$	(-s,-s)	$(-\alpha,\beta)$	(0,1)	(α,β)	(s,-s)	$(-\beta,-\alpha)$	(-1,0)	$(-\beta,\alpha)$	(s,s)	$(\alpha,-\beta)$	(0,-1)	$(-\alpha,-\beta)$	(-s,s)
(1,0)	(0,-1)	(-1,0)	(1,0)	(0,1)	(0,-1)	(-1,0)	(0,1)	(1,0)	(0,-1)	(1,0)	(0,1)	(1,0)	(0,-1)	(0,1)
(1,0)	$(-\beta,-\alpha)$	(-s,s)	(α,β)	(0,-1)	$(-\alpha,\beta)$	(s,s)	(0,-1)	(1,0)	$(\beta,-\alpha)$	(s,-s)	$(-\alpha,-\beta)$	(0,1)	$(\alpha,-\beta)$	(-s,-s)
(1,0)	(-s,-s)	(0,1)	(s,-s)	(-1,0)	(s,s)	(0,-1)	(1,0)	(-s,s)	(0,1)	(s,-s)	(-1,0)	(s,s)	(0,-1)	(-s,s)
(1,0)	$(-\alpha,-\beta)$	(s,s)	$(-\beta,-\alpha)$	(0,1)	(-s,s)	$(\alpha,-\beta)$	(-1,0)	(α,β)	(-s,-s)	(β,α)	(0,-1)	$(-\beta,\alpha)$	(s,-s)	$(-\alpha,\beta)$
(1,0)	(-1,0)	(1,0)	(-1,0)	(1,0)	(-1,0)	(1,0)	(-1,0)	(1,0)	(-1,0)	(1,0)	(-1,0)	(1,0)	(-1,0)	(-1,0)
(1,0)	$(-\alpha,\beta)$	(s,-s)	$(-\beta,\alpha)$	(0,-1)	(α,β)	(-s,-s)	(1,0)	$(-\alpha,-\beta)$	(-s,s)	(0,1)	$(\beta,-\alpha)$	(0,-1)	$(\alpha,-\beta)$	$(\alpha,-\beta)$
(1,0)	(-s,s)	(0,1)	(-s,s)	(0,-1)	(s,-s)	(1,2)	(-s,s)	(0,-1)	(s,s)	(0,1)	(-s,-s)	(1,0)	(s,-s)	(-s,-s)
(1,0)	(-1,0)	(-s,-s)	(0,1)	(-1,0)	(s,-s)	(-1,0)	(0,1)	(-1,0)	(s,s)	(0,-1)	(-s,s)	(1,0)	(-s,s)	(-s,s)
(1,0)	$(\beta,1)$	(-1,0)	$(\beta,-\alpha)$	(0,1)	$(-\alpha,\beta)$	(0,-1)	(1,0)	$(\beta,-\alpha)$	(s,-s)	(0,1)	$(-\alpha,-\beta)$	(0,1)	$(-\beta,-\alpha)$	$(-\beta,\alpha)$
(1,0)	(0,1)	(1,0)	(0,-1)	(1,0)	(0,1)	(-1,0)	(1,0)	(0,1)	(-1,0)	(1,0)	(0,-1)	(1,0)	(0,1)	(0,-1)
(1,0)	(α,β)	(-s,s)	$(\alpha,-\beta)$	(s,s)	$(-\beta,\alpha)$	(-1,0)	$(\alpha,-\beta)$	(-1,0)	$(\alpha,-\beta)$	(s,-s)	(α,β)	(0,-1)	$(-\alpha,\beta)$	$(\beta,-\alpha)$
(1,0)	(s,s)	(0,1)	(-s,-s)	(0,-1)	(s,-s)	(-1,0)	(s,s)	(1,0)	(s,-s)	(0,1)	(s,s)	(1,0)	(-s,-s)	(s,-s)
(1,0)	(a,b)	(s,s)	(β,α)	(0,1)	$(-\beta,\alpha)$	(-s,s)	$(-\alpha,\beta)$	(-1,0)	$(-\alpha,-\beta)$	(-s,-s)	$(-\beta,-\alpha)$	(0,-1)	$(\beta,-\alpha)$	$(\alpha,-\beta)$

Since N=5 will be used to accomplish a transform of the order of 4660 for the SAR range processing, $\hat{B}_5$ will be examined in more detail. First note in Figure C3 that only row/column terms with index values greater than Ø require multiplies. Moreover, note that if $\hat{B}_5$ acts on a real input vector, $y_4 = y_1^*$ and $y_3 = y_2^*$. Since a complex input can be treated as a pair of real inputs*, it is only necessary to show how to compute y_1 and y_2 for a real input vector. Note that for the complex input vector X

$$y_1 = (X_o + cX_1 - eX_2 - eX_3 + cX_4) + (-dX_1 - fX_2 + fX_3 + dX_4)i \quad \text{and}$$

$$y_2 = (X_o - eX_1 + cX_2 + cX_3 - eX_4) + (-fX_1 + dX_2 - dX_3 + fX_4)i$$

where $i = \sqrt{-1}$

$c = \cos 72^o$ $\quad$ $d = \sin 72^o$

$e = \cos 36^o$ $\quad$ $f = \sin 36^o$

* To produce the DFT of the complex input vector, both the real part of the input and the imaginary part are treated as 2 distinct real input vectors. After the DFT's of both real input vectors have been generated, the 2 transforms are recombined so as to develop the DFT of the complex input vector.

To minimize multiplies needed to compute y_1 and y_2, first assume that the auxiliary calculations P,Q,R,S,T are performed as shown below:

$$P = ((x_1+x_4)-(x_2+x_3))*((c+e)/2)$$

$$Q = ((x_1+x_4)+(x_2+x_3))*(-(c-e)/2)$$

$$R = ((x_1-x_4)+(x_2-x_3))*(-(f+d))$$

$$S = (x_1-x_4)*(-f)$$

$$T = (x_2-x_3)*d \qquad \text{where}$$

$(c+e)/2=.55901699=\sqrt{5}/4$, $(-(c-e)/2)=1/4$, $(-f+d))=1.5388417$
$-f=-.58778525$, and $d=.9510565$. Then, for the real part of $\tilde{x}$, $\tilde{y}_1,\tilde{y}_4=\text{REAL}(y_1)=(\tilde{x}_0+Q)+P=c(\tilde{x}_1+\tilde{x}_4)-e(\tilde{x}_2+\tilde{x}_3)$

$\tilde{y}_2,\tilde{y}_3=\text{REAL}(y_2)=(\tilde{x}_0+Q)-P=-e(\tilde{x}_1+\tilde{x}_4)+c(\tilde{x}_2+\tilde{x}_3)$

$\tilde{\tilde{y}}_2,-\tilde{\tilde{y}}_3=\text{IMAGINARY}(y_2)=S+T=\quad -f(x_1-x_4)+d\ (x_2-x_3)$

$\tilde{\tilde{y}}_1,-\tilde{\tilde{y}}_4=\text{IMAGINARY}(y_1)=R-(S-T)=-d\ (x_1-x_4)-f\ (x_2-x_3)$ A like result is found when treating the imaginaly part of X. Thus, the processing flow diagram for a complex vector X whose components are given by $(\tilde{x}_i,\tilde{\tilde{x}}_i)$ where $i=0,..,4$ is that shown in Figure C4.

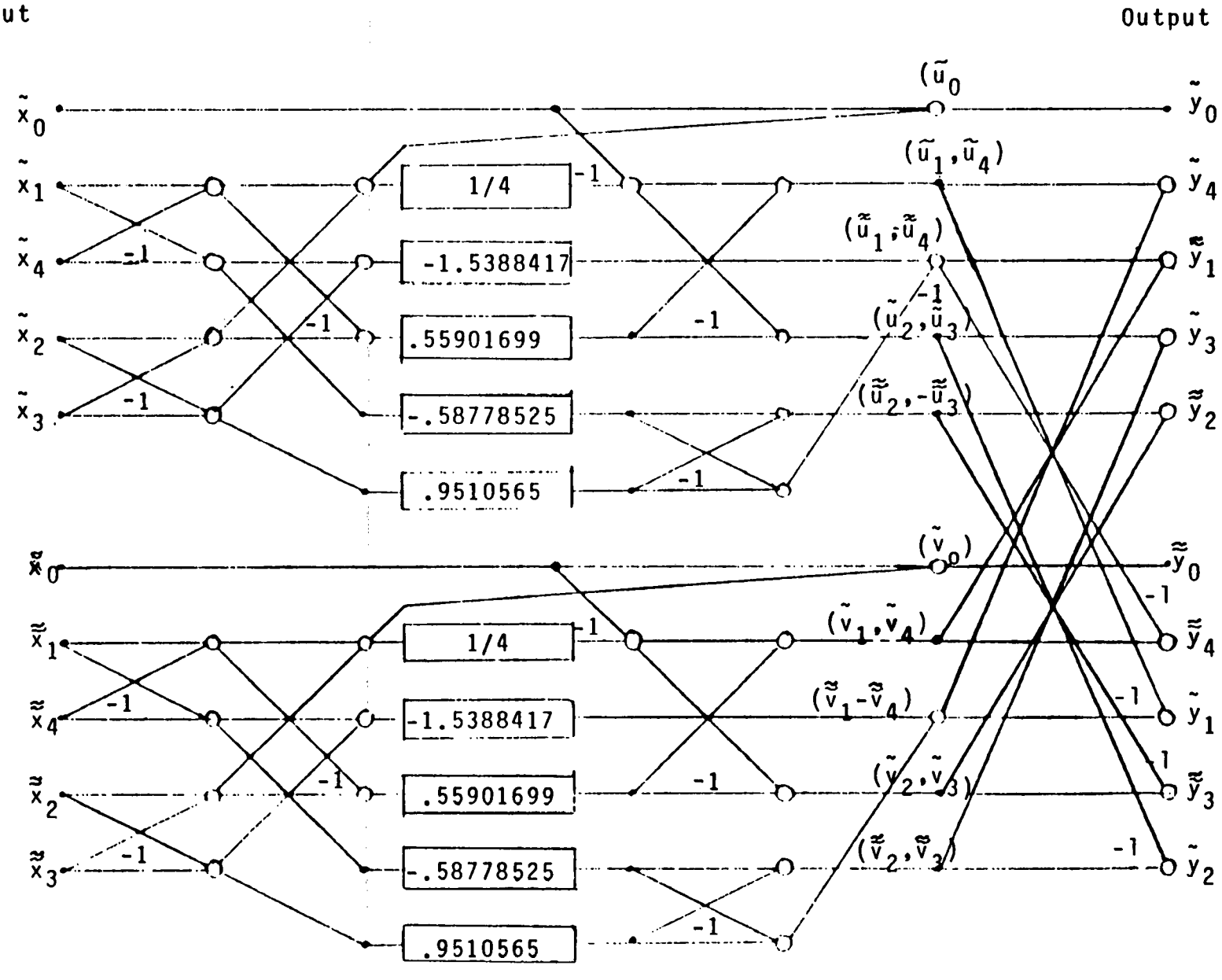

FIGURE C4 - Five Point DFT Processing Flow for MPP

APPENDIX D

DESIGN OF A MASSIVELY PARALLEL PROCESSOR

Kenneth E. Batcher
Goodyear Aerospace Corporation, Akron, Ohio

Abstract -The massively parallel processor (MPP) system is designed to process satellite imagery at high rates. A large number (16,384) of processing elements (PE's) are configured in a square array. For optimum performance on operands of arbitrary length, processing is performed in a bit-serial manner. On 8-bit integer data, addition can occur at 6553 million operations per second (MOPS) and multiplication at 1861 MOPS. On 32-bit floating-point data, addition can occur at 430 MOPS and multiplication at 216 MOPS.

Index Terms -Array processing, bit-slice processing, computer architecture, image processing, parallel processing, satellite imagery.

INTRODUCTION

In this decade, NASA will orbit imaging sensors that can generate data at rates up to 10^{13} bits per day. A variety of image processing tasks such as geometric correction, correlation, image registration, feature selection, multispectral classification, and area measurement are required to extract useful information from this mass of data. The expected workload is between 10^9 and 10^{10} operations per second.

In 1971 NASA Goddard Space Flight Center initiated a program to develop ultra high-speed image processing systems capable of processing this workload. These systems use thousands of processing elements (PE's) operating simultaneously (massive parallelism) to achieve their speed. They exploit the fact that the typical satellite image contains millions of picture elements (pixels) that can generally be processed at the same time.

In December 1979 NASA Goddard awarded a contract to Goodyear Aerospace to construct a massively parallel processor (MPP) to be delivered in the first quarter of 1982. The basic elements of the MPP architecture were developed at NASA Goddard. This correspondence presents the design of the MPP system. The major components are shown in Fig. 1. The array unit (ARU) processes arrays of data at high speed and is controlled by the array control unit (ACU), which also performs scalar arithmetic. The program and data management unit (PDMU) controls the overall flow of data and programs through the system and handles certain ancillary tasks such as program development and diagnostics. The staging memories buffer and reorder data between the ARU, PDMU, and external (host) computer.

ARRAY UNIT

Logically, the array unit (ARU) contains 16,384 processing elements (PE's) organized as a 128 by 128 square. Physically, the ARU has an extra 128 by 1 rectangle of PE's that is used to reconfigure the ARU when a PE fault is detected. The PE's are bit-serial processors for efficiently processing operands of any length. The basic clock rate is 10 MHz. With 16,384 PE's operating in parallel, the ARU has a very high processing speed (see Table 1). Despite the bit-serial nature of the PE's, even the floating-point speeds compare favorably with several fast number crunchers.

TABLE I - SPEED OF TYPICAL OPERATIONS

Operations	Execution Speed*
Addition of Arrays	
8-bit integers (9-bit sum)	6553
12-bit integers (13-bit sum)	4428
32-bit floating-point numbers	430
Multiplication of Arrays (Element-by-Element)	
8-bit integers (16-bit product)	1861
12-bit integers (24-bit product)	910
32-bit floating-point numbers	216
Multiplication of Array by Scalar	
8-bit integers (16-bit product)	2340
12-bit integers (24-bit product)	1260
32-bit floating-point numbers	373

*Million operations per second

Routing Topology

Each PE in the 128 by 128 square communicates with its nearest neighbor; up, down, right, and left—a topology similar to Illiac IV and some other array processors. Alternative routing topologies such as the flip network (1) or one of its equivalents (2) were investigated. They have the ability to shift data over many PE's in one step and allow data to be accessed in many different directions (3). Certain paths in the alternative topologies have long runs that complicate their layout and limit their cycle rate. When the number of PE's interconnected is only 256 as in the STARAN® computer, this is no problem; when 16,384 PE's are interconnected, it is a severe problem.

Most of the expected workload does not use the routing flexibility of the alternative topologies. The ability to access data in different directions is important when arrays of data are input and output; it can be used to reorient the arrays between the bit-plane format of the ARU and the pixel format of the outside world.

These considerations lead to the conclusion that the ARU should have a two-dimensional nearest neighbor routing topology such as Illiac IV since it is easy to implement and matches the two-dimensional nature of satellite imagery. The problem of reformatting I/O data is best handled in a staging memory between the ARU and the outside world.

Around the edges of the 128 by 128 array of PE's the edges can be left open (e.g., a row of zeros can be entered along the left edge when routing data to the right) or the opposite edges can be connected. Cases were found where open edges were preferred and other cases where connected edges were preferred. It was decided to make edge-connectivity a programmable function. A topology-register in the array control unit defines the connections between opposite edges of the PE array. The top and bottom edges can either be connected or left open. The connectivity between the left and right edges has four states: open (no connection), cylindrical (connect the left PE of each row to the right PE of the same row), open spiral (for $1 \le n \le 127$, connect the left PE of row n to the right PE of row n-1), and closed spiral (like the open spiral, but also connect the left PE of row 0 to the right PE of row 127).

The spiral modes connect the 16,384 PE's together in one long linear array. One can pack several linear arrays of odd sizes (e.g., lines with thousands of image pixels per line) in the ARU and process them in parallel.

Redundancy

The ARU includes some redundancy so that a faulty PE can be bypassed. Redundancy can be added to a two-dimensional array of PE's by adding an extra column (or row) of PE's and inserting bypass gates in the routing network. When a faulty PE is discovered, one disables the whole column containing the faulty PE and joins the columns on either side of it with the bypass gates.

The PE's in the ARU are implemented with two-row by four-column VLSI chips; thus, it is more convenient to add four redundant columns of PE's and bypass four columns at a time. The PE array has 128 rows and 132 columns. It is divided into 33 groups, with each group containing 128 rows and four columns of PE's. Each group has an independent group-disable control line from the ACU. When a group is disabled, all its outputs are disabled and the groups on either side of it are joined together with 128 bypass gates in the routing network.

When there is no faulty PE, an arbitrary group is disabled so that the size of the logical array is always 128 by 128. Application programs are not aware of which group is disabled and need not be modified when the disabled group is changed. They always use the logical address of a PE to access PE dependent data. The logical address of a PE is a pair of 7-bit numbers X and Y showing its position in the logical array of enabled PE's. A simple routine executed in 27 μs will load the memory of each PE with its logical address.

When a faulty PE is discovered, its data cannot be trusted so the normal error recovery procedure is to reconfigure the ARU to disable the column containing the fault and then to restart the application program from the last checkpoint or from the beginning.

Bit-Serial Processing

The data arrays being processed have a wide range of element lengths. A spectral band of an input pixel may have a resolution of 6 to 12 bits. Intermediate results can have any length from 6 to more than 30 bits. Single-bit flag arrays are generated when pixels are classified. Some computations may be performed in floating point. Thus, the PE's should be able to process operands of any length efficiently.

Conventional computers typically use bit-parallel arithmetic units with certain fixed-word lengths such as 8, 16, or 32 bits. Operands of odd lengths are extended to fit the standard word sizes of the machine. Some of the hardware in the memory and the arithme-

tic unit is wasted storing and processing the extensions.

Bit-serial processors process operands bit by bit and can handle operands of any length without any wasted hardware. Their slower speed can be counteracted by using a multitude of them and processing many operands in parallel.

There is a wide variety of operand lengths and a prevalence of low-precision operands in the expected workload. Thus, bit-serial processors are more efficient in the use of hardware than bit-parallel processors.

Processing Elements

The initial MPP design had PE's using downshifting binary counters for arithmetic (4), (6), (7). The PE design was modified to use a full adder and shift register combination for arithmetic. The modified design performs the basic arithmetic operations faster. Each of the PE's has six 1-bit registers (*A*, *B*, *C*, *G*, *P*, and *S*), a shift register with a programmable length, a random-access memory, a data bus (*D*), a full adder, and some combinatorial logic (see Fig. 2). The basic cycle time of the PE is 100 ns.

Logic and Routing: The *P*-register is used for logic and routing operations. A logic operation combines the state of the *P*-register and the state of the data bus (*D*) to form the new state of the *P*-register. All 16 Boolean functions of the two variables *P* and *D* are implemented. A routing operation shifts the state of the *P*-register into the *P*-register of a neighboring PE (up, down, right, or left).

Arithmetic: The full adder, shift register, and registers *A*, *B*, and *C* are used for bit-serial arithmetic operations. To add two operands, the bits of one operand are put into the *A*-register, one at a time, least significant bit (LSB) first; corresponding bits of the other operand are put into the *P*-register; the full adder adds the bits in *A* and *P* to the carry bits in the *C*-register to form the sum and carry bits; each carry bit is stored in *C* to be added in the next cycle; and each sum bit is stored in *B*. The sum formed in *B* can be stored in the random-access memory and/or in the shift register. Two's complement subtraction is performed by adding the one's complement of the operand in *P* to the operand in *A* and setting the initial carry bit in *C* to 1 instead of 0.

Multiplication is a series of addition steps where the partial product is recirculated through the shift register and registers *A* and *B*. Appropriate multiples of the multiplicand are formed in *P* and added to the partial product as it recirculates. Division is performed with a nonrestoring division algorithm. The partial dividend is recirculated through the shift register and registers *A* and *B* while the divisor or its complement is formed in *P* and added to it.

Floating-point addition compares exponents; places the fraction of the operand with the least exponent in the shift register; shifts it to the right to align it with the other fraction; adds the other fraction to the shift register; and normalizes the sum. Floating-point multiplication is a multiplication of the fractions, a normalization of the product, and an addition of the exponents.

Masking: The *G*-register can hold a mask bit that can control the activity of the PE. Unmasked logic, routing, and arithmetic operations are performed in all PE's. Masked operations are only performed in those PE's where the *G*-register equals 1.

Several operations may be combined in one 100 ns instruction. Logic and routing operations are masked independently of arithmetic operations so one can combine a masked routing operation with an unmasked arithmetic operation or vice versa. This feature proves to be quite useful in a number of algorithms.

Storage: The random-access memory stores 1024 bits per PE. Standard RAM integrated circuits are used to make it easy to expand storage as advances occur in solid-state memory technology. The ACU generates 16-bit addresses so ARU storage can be expanded to 65,536 bits per PE. Thus, the initial complement of 2 Mbytes of ARU storage can be expanded sixty-fourfold if technology allows it.

Parity error detection is used to find memory faults. A parity bit is added to the eight data bits of each 2 by 4 subarray of PE's. Parity bits are generated and stored for each memory write cycle and checked when the memories are read. A parity error sets an error flip-flop associated with the 2 by 4 subarray. A tree of logic element gives the ACU an inclusive-or of all error flip-flops (after some delay). By operating the group-disable control lines, the ACU can locate the group containing the error and disable it.

Sum-Or Tree: The data bus states of all enabled PE's are combined in a tree of inclusive-or elements called the sum-or tree. The output of this tree is fed to the ACU and used in certain operations such as finding the maximum or minimum value of an array in the ARU.

Input/Output: The *S*-register is used to input and output ARU data. While the PE's are processing data in the random-access memories, columns of input data are shifted into the left side of the ARU (Fig. 1) and through the *S*-registers (Fig. 2) until a plane of 16,384 bits is loaded. The input plane is then stored in the random-access memories in one 100 ns cycle by inter-

rupting the processing momentarily in all PE's and moving the S-register values to the memory elements. Planes of data are output by moving them from the memory elements to the S-registers and then shifting them out column by column through the right side of the ARU. The shift rate is 10 MHz; thus, up to 160 Mbytes/s can be transferred through the ARU I/O ports. Processing is interrupted for 100 ns for each bit plane of 16,384 bits transferred—less than 1 percent of the time.

Packaging

Standard 4 by 1024-bit RAM elements are used for the PE memories. All other components of a 2 by 4 subarray of PE's are packaged on a custom VLSI CMOS/SOS chip. The VLSI chip also contains the parity tree and the bypass gates for the subarray.

Each 8-1/2 in. by 14 in. printed circuit board contains 192 PE's in a 16 by 12 array. A board contains 24 VLSI chips, 54 memory elements, and some support circuitry. Eleven boards make up an array slice of 16 by 132 PE's.

Eight array slices (88 boards) make up the ARU. Eight other boards contain the topology switches, control fan out, and other support circuitry. The 96 boards of the ARU are packaged in one cabinet (the leftmost cabinet in Fig. 3). Forced-air cooling is used.

ARRAY CONTROL UNIT

Like the control units of other parallel processors, the array control unit (ACU) performs scalar arithmetic and controls the PE's. It has three sections that operate in parallel (see Fig. 4): PE control, I/O control, and main control. PE control performs all array arithmetic of the application program. I/O control manages the flow of data in and out of the ARU. Main control performs all scalar arithmetic of the application program. This arrangement allows array arithmetic, scalar arithmetic, and input/output to be overlapped for minimum execution time.

PE Control

PE control generates all ARU control signals except those associated with I/O. It contains a 64-bit common register to hold scalars and eight 16-bit index registers to hold the addresses of bit planes in the PE memory elements, to count loop executions, and to hold the index of a bit in the common register. PE control reads 64-bit-wide microinstructions from PE control memory. Most instructions are read and executed in 100 ns. One instruction can perform several PE operations, manipulate any number of index registers, and branch conditionally. This reduces overhead significantly so that little, if any, PE processing power is wasted.

PE control memory contains a number of system routines and user-written routines to operate on arrays of data in the ARU. The routines include both array-to-array and scalar-to-array arithmetic operations. A queue between PE control and main control queues up to 7 calls to the PE control routines. Each call contains up to 8 initial index-register values and up to 64 bits of scalar information. Some routines extract scalar information from the ARU (such as a maximum value) and return it to main control.

I/O Control

I/O control shifts the ARU S-registers, manages the flow of information in and out of the ARU ports, and interrupts PE control momentarily to transfer data between the S-registers and buffer areas in the PE memory elements. Once initiated by main control or the PDMU, I/O control can chain through a number of I/O commands. It reads the commands from main control memory.

Main Control

Main control is a fast scalar processor. It reads and executes the application program in the main control memory. It performs all scalar arithmetic itself and places all array arithmetic operations on the PE control call queue. It contains 16 general purpose registers, three registers to control the ARU group-disable lines, 13 registers associated with the call queue, 12 registers to receive scalars from PE control, and six registers to monitor and control the status of PE control, I/O control, and the ARU.

PROGRAM AND DATA MANAGEMENT UNIT

The program and data management unit (PDMU) controls the overall flow of programs and data in the system (Fig. 1). Control is from an alphanumeric terminal. The PDMU is a minicomputer (DEC PDP-11) with custom interfaces to the ACU control memories and registers and to the staging memories. The operat-

ing system is DEC's RSX-11M real-time multiprogramming system.

The PDMU also executes the MPP program-development software package. The package includes a PE control assembler to develop array processing routines for PE control, a main assembler to develop application programs executing in main control, a linker to form load modules for the ACU, and a control and debug module that loads programs into the ACU, controls their execution, and facilitates debugging. This package is written in Fortran for easy movement to the host computer.

STAGING MEMORIES

The staging memories reside between the wide I/O ports of the ARU and the PDMU. They also have a port to an external (host) computer. Besides acting as buffers for ARU data being input and output, the memories reorder arrays of data.

Satellite imagery is normally stored in pixel order in the PDMU and other conventional computers. That is, line 1 pixel 1 followed by line 1 pixel 2, etc., followed by the pixels of line 2, line 3, etc. The imagery might be band-interleaved (all spectral bands of a pixel stored together) or band-sequential (band 1 of all pixels followed by band 2 of all pixels, etc.).

Arrays of data are transferred through the ARU ports in bit-sequential order. That is, the most (or least) significant bit of 16,384 elements followed by the next bit of 16,384 elements, etc. Reordering is required to fit the normal order of satellite imagery in the PDMU or the host. Thus the staging memories are given a reordering capability.

The staging memories are packaged together in a large multidimensional-access (MDA) or corner-turning memory. Items of data flow through a substager which is a smaller MDA memory. Input data items from the ARU, PDMU, or host are reformatted in the substager into patches which are sent to the large staging memory. Output data patches from the large staging memory are reformatted in the substager for transmission to the ARU, PDMU, or host.

The large staging memory uses 1280 dynamic RAM integrated circuits for data storage and 384 RAM's for error-correcting-code (ECC) storage. (A 6-bit ECC is added to each 20-bit word.) Initially, the boards will be populated with 16K bit RAM's for a capacity of 2.5 Mbytes. Later, as memory technology advances, the 16K bit RAM's can be replaced with 64K bit RAM's or 256K bit RAM's to increase the capacity of 10 Mbytes or 40 Mbytes.

The substager can access the main stager at a 320 Mbyte per second rate (thirty-two 20-bit words every 250 ns). The accesses can be spread across the main stager in a variety of ways. Patches of data in various orientations can be read or written conveniently.

The substager has a smaller memory with 1-bit words. It assembles input data into patches for the main staging memory and disassembles patches of data from the main staging memory for output.

The main staging memory and the substager are controlled by a control unit that can be programmed to input and output imagery in a wide variety of formats.

HOST INTERFACE

The MPP to be delivered to NASA will use a DEC VAX-11/780 for a host computer. The staging memories of the MPP are connected to a DR-780 high-speed user interface of the VAX-11/780. Imagery can be transferred over this path at the rate of the DR-780 (at least 6 Mbytes/s). To allow control of the MPP by the host, the custom interface of the MPP is switched from the PDMU to the host. The switching is simplified by the fact that both the PDMU (a DEC PDP-11) and the host (a DEC VAX-11/780) have a DEC UNIBUS. The transfer of system software to the host is simplified by writing much of it in Fortran and using the compatibility mode of VAX to execute those portions written in PDP-11 code.

CONCLUSIONS

The massively parallel processor is designed to process satellite imagery at high rates. Its high-processing speed, large memory capacity, and I/O reformatting capabilities should make it useful in other applications. Preliminary studies indicate that the MPP can support such diverse application areas as general image processing, weather simulation, aerodynamic studies, radar processing, reactor diffusion analysis, and computer-image generation.

The modular structure of the MPP allows it to be scaled up or down for different applications. The number of processing elements in the ARU can be adjusted to support different processing rates. The sizes of the ARU and staging memories are also adjustable. Host computers other than the VAX-11/780 can be accommodated. The PDMU functions can be absorbed by the host or alternatively, the PDMU can act as the host (since the PDMU is in the PDP-11 and VAX family, a wide variety of PDMU capacities and configurations is feasible).

As part of its ongoing program to develop spaceborne image processors, NASA-Goddard is pursuing the design of a miniaturized version of the MPP (5).

REFERENCES

(1) K. E. Batcher, "The Flip Network in STARAN," in *1976 Proc. Int. Conf. Parallel Processing*, pp. 65-71.

(2) H. J. Siegel and S.D. Smith, "Study of Multistage SIMD Interconnection Networks," in *Proc. 5th Annual Symp. Computer Architecture*, April 1978, pp. 223-229.

(3) K. E. Batcher, "The Multi-dimensional-access Memory in STARAN," *IEEE Trans. Computer*, vol. C-26, pp. 174-177, Feb. 1977.

(4) L.W. Fung, *A Massively Parallel Processing Computer: High-Speed Computer and Algorithm Organization*, D. J. Kuck *et al*, Ed. New York: Academic, 1977, pp. 203-204.

(5) D. H. Schaefer, "Massively Parallel Information Processing Systems for Space Applications," presented at AIAA Computer Aerospace Conf. II, Oct. 1979.

(6) L. W. Fung, "MPPC: A Massively Parallel Processing Computer," Goddard Space Flight Center, Greenbelt, MD, GSFC Image Systems Section Report, Sept. 1976.

(7) Request for Proposal, RFP GSFC-5-45191/254 (Appendix A).

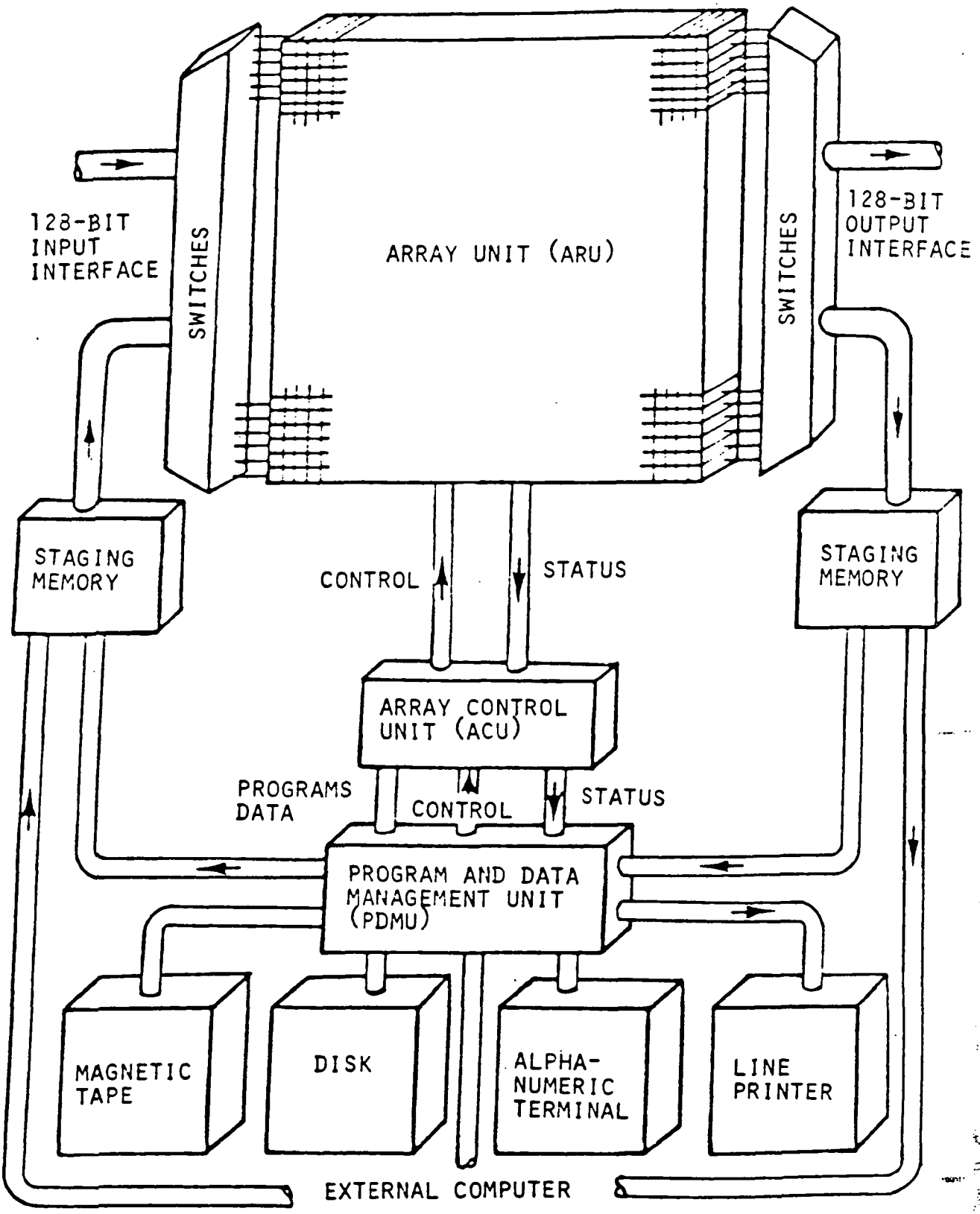

Figure 1 - Block Diagram of the Massively Parallel Processor (MPP)

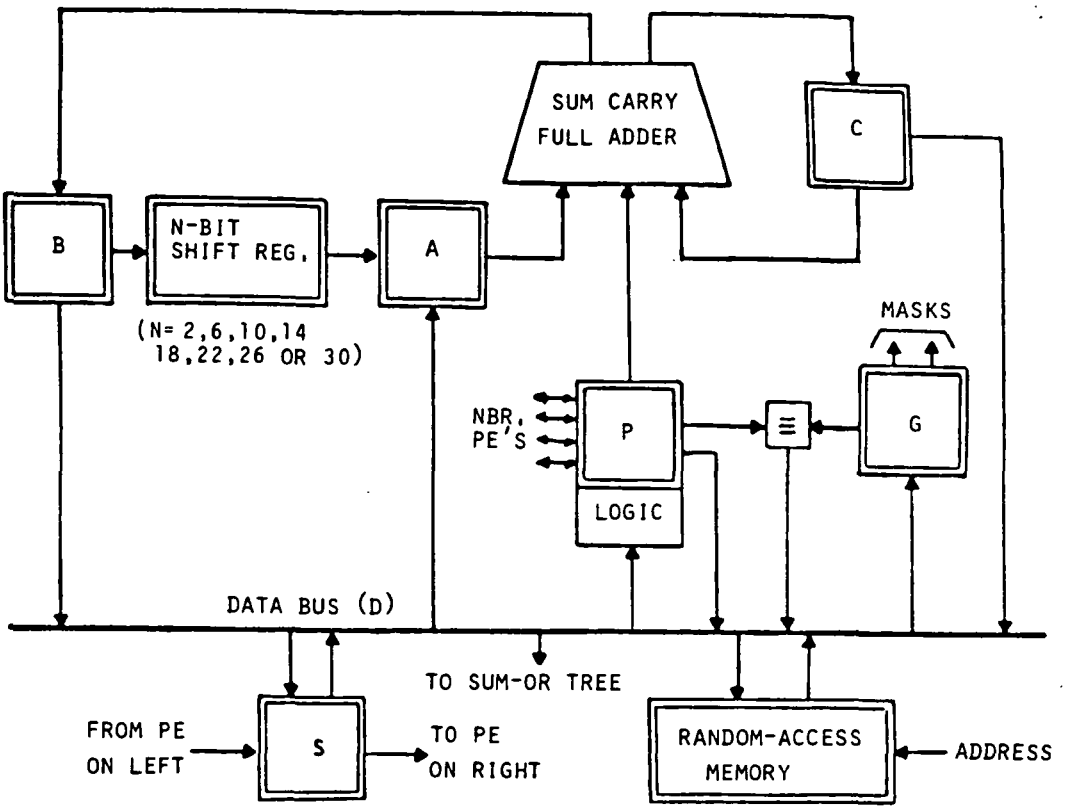

Figure 2 - One Processing Element

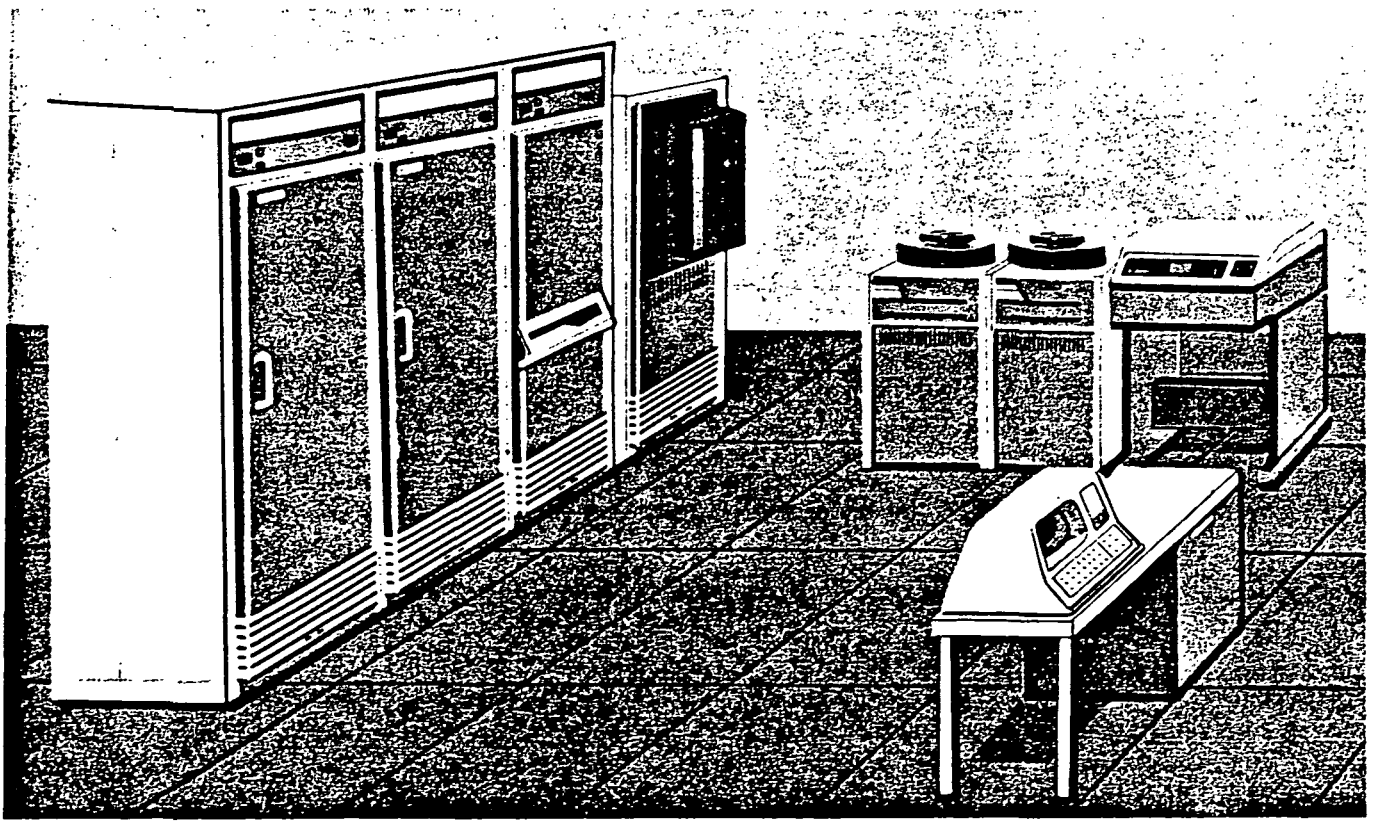

Figure 3 - MPP Physical Configuration

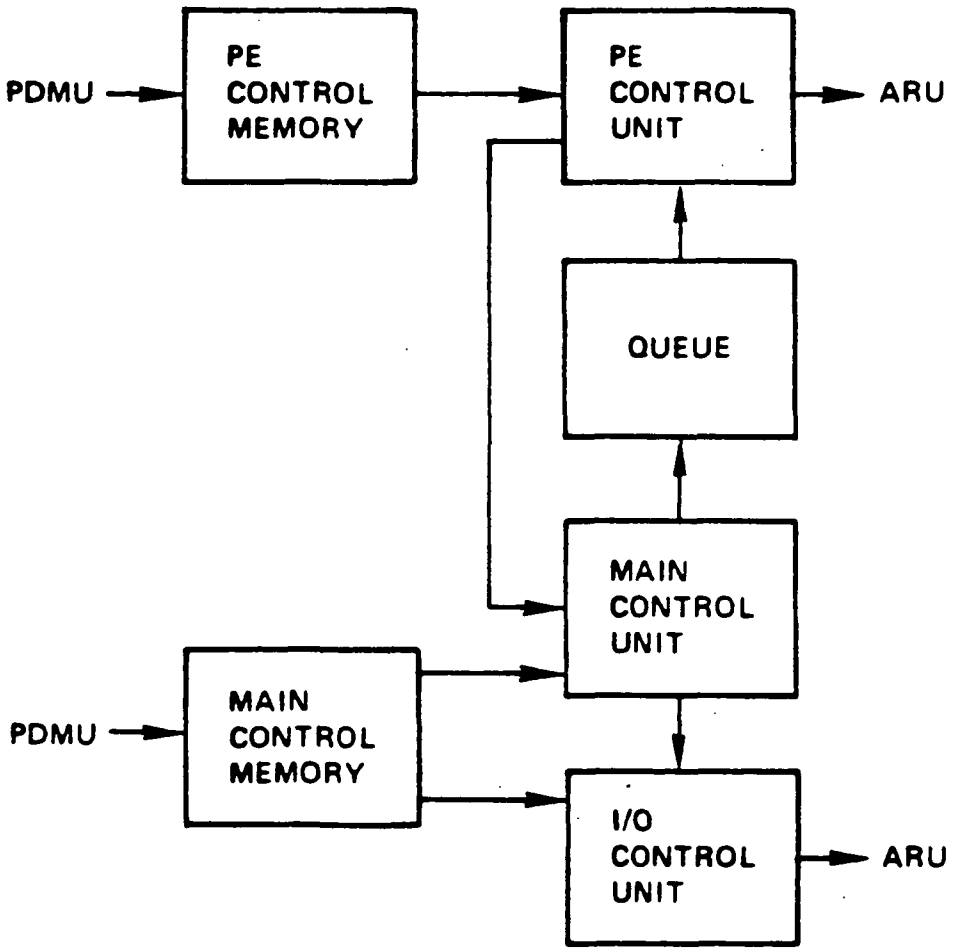

Figure 4 - Block Diagram of Array Control Unit (ACU)