

AD-A130 655

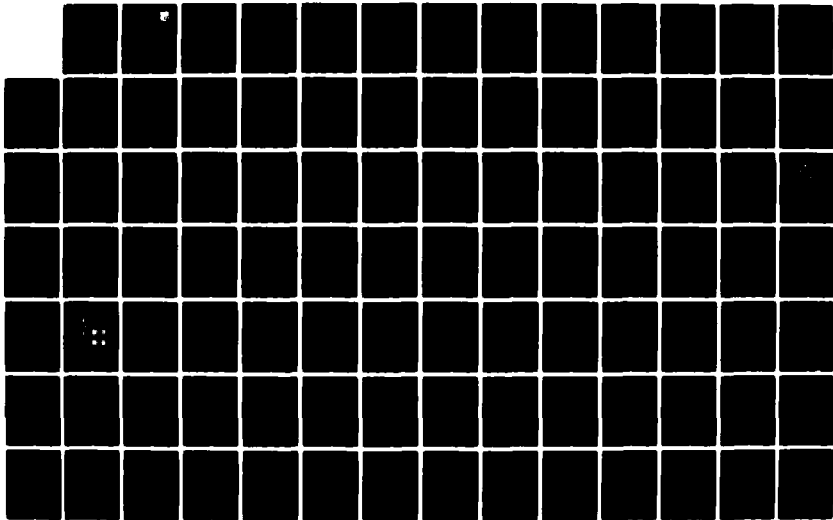
MULTIMODE RADAR SIGNAL PROCESSOR INTEGRATION FACILITY
(U) AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON
AFB OH J N HORN ET AL. MAY 83 AFWAL-TR-83-1045

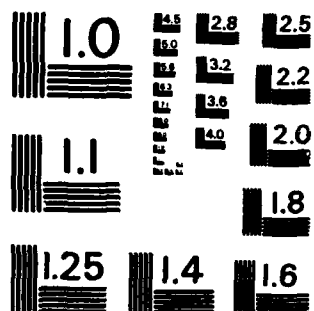
1/2

UNCLASSIFIED

F/G 17/9

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFWAL-TR-83-1045

AD A130655



MULTIMODE RADAR SIGNAL PROCESSOR
INTEGRATION FACILITY

John N. Horn
2Lt Gregory A. Frascadore
Byron R. Stephens

May 1983

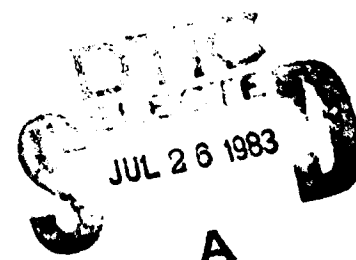
Interim Report for Period February 1981 - December 1982

Approved for public release; distribution unlimited

DTIC FILE COPY

Copy available to DTIC does not
permit fully legible reproduction

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433



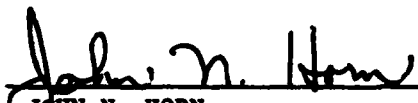
83 07 26 150

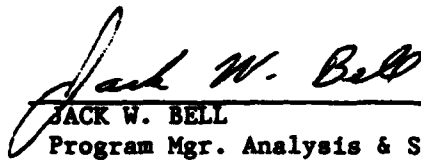
NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

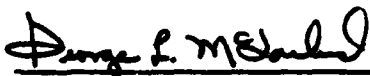
This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.


JOHN N. HORN
Project Engineer
Analysis and Signal Processing


JACK W. BELL
Program Mgr. Analysis & Sig Processing
Mission Avionics Division

FOR THE COMMANDER


GEORGE L. McFARLAND, Chief
Radar Branch
Mission Avionics Division

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AARM-3 W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DTIC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-83-1045	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MULTIMODE RADAR SIGNAL PROCESSOR INTEGRATION FACILITY		5. TYPE OF REPORT & PERIOD COVERED INTERIM REPORT FOR PERIOD 3 FEB 1981 - 15 DEC 82
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) JOHN N. HORN 2LT GREGORY A. FRASCADORE BYRON R. STEPHENS		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AVIONICS LABORATORY (AFWAL/AARM-3) AIR FORCE WRIGHT AERONAUTICAL LABORATORIES (AFSC) WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 7622 05 11
11. CONTROLLING OFFICE NAME AND ADDRESS AVIONICS LABORATORY (AFWAL/AARM-3) AIR FORCE WRIGHT AERONAUTICAL LABORATORIES (AFSC) WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433		12. REPORT DATE May 1983
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 99
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Radar Signal Processing, Digital Computer.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This effort is centered on the Multimode Radar Signal Processor (MRSP), which is a high speed prototype signal processor based on the Westinghouse PSP-X design. The basic objective is to gain familiarity and programming proficiency with this relatively complex system, and ultimately apply its high speed capabilities to practical radar signal processing problems. Brief descriptions of the MRSP hardware and software configurations are included for reader continuity. The major portion of the report describes specific		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 68 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

activities related to MRSP operations. Primary among these was the development of a software package to integrate a video output display system to the MRSP. Additionally, several representative programs were written to demonstrate the high speed signal processing capabilities of the MRSP as a complete system. Finally, several significant improvements were incorporated into the original MRSP support software package provided by the contractor.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

FOREWORD

The work described in this report was performed under AFWAL/AARM-3 in-house work unit 7622-05-11, covering the period 3 February 1981 through 15 December 1982. The work was performed by John N. Horn, 2Lt Gregory A. Frascadore, and Byron R. Stephens, all of AFWAL/AARM-3, Wright-Patterson AFB OH. This report was submitted by the authors in March 1983.

The authors wish to thank Mr. Richard C. Vanderburg for his knowledgable advice and helpful suggestions.



✓

A 53
2

TABLE OF CONTENTS

SECTION		PAGE
I	INTRODUCTION	1
	1. General Discussion	1
	2. MSRP Overview	1
	3. Program Milestones	3
II	MSRP HARDWARE DESCRIPTION	6
	1. Overall Configuration	6
	2. System Components	9
	a. Physical Description	9
	b. Vector Arithmetic Processor (VAP)	9
	c. Control Arithmetic Processor (CAP)	10
	d. Input/Output Controller (IOC) And Bulk Memory	10
	e. Direct Memory Access (DMA) Channel Bus Controller	11
	f. FIFO Interface	11
	g. Differential Address Generator (DAG)	12
	h. Console Intelligence Unit (CIU)	13
	i. Ramtek Display System	13
	j. VAX 11/750 Host Computer	14
III	MRSP SYSTEM SOFTWARE DESCRIPTION	15
	1. General Description	15
	2. Support Software Overview	15
	3. CAP Support Package	17
	4. Host I/O Driver	18
	5. VAP Simulator	20
	6. System Diagnostics	21
	7. Host Processor Operating System	21

TABLE OF CONTENTS (Cont'd)

SECTION	PAGE
IV MSRP System Operations	23
1. RAMTEK Display Integration	23
a. General Approach	27
b. Command File Descriptions	28
(1) WTEXT	29
(2) WVECTOR	32
(3) WPLOT	33
(4) WIMAGEU And WIMAGEL	35
(5) WLOOKUP	36
(6) WCURSOR	38
(7) RAMCLEAR	39
(8) NEGATE	40
c. SUMMARY	40
2. Demonstration Program	41
a. Function Description	42
b. Implementation	45
c. Timing Considerations	50
3. Program for Verification of DAG Operations	51
a. Functional Description	52
b. Program Operation	52
c. DAG Requirement	53
d. Considerations	54
4. Support Software Modifications	54
V CONCLUSIONS AND FUTURE ACTIVITIES	56
REFERENCES	93

TABLE OF CONTENTS (Cont'd)

SECTION	PAGE
APPENDIX A SAMPLE PROGRAM LISTINGS	59
APPENDIX B FASTBIN USERS MANUAL AND INSTALLATION GUIDE	75
APPENDIX C CAS COMMAND FILE	89

LIST OF ILLUSTRATIONS

FIGURE		PAGE
1	MRSP Configuration	7
2	Basic RAMSTEK Instruction Set	25
3	Instruction Parameter Format	26
4	Standard Text Character Fonts	30
5	A Pulse Function and Fourier Transform	43
6	Sampled Fourier Transform Pair	44
7	Three-Dimensional Representations	46

SECTION I

INTRODUCTION

1. GENERAL DISCUSSION

This is an interim technical report for the MRSP Integration Facility Program. The program is being conducted as an in-house effort under AFWAL/AARM Work Unit Number 76220511. This report covers the period from February 1981 through December 1982. The effort is centered on the Multi-mode Radar Signal Processor (MRSP), which is a high speed prototype processor based on the Westinghouse PSP-X System. The overall objective is to "wring out" the MRSP and gain experience and programming proficiency. This effort is intended to serve as an interim preparation for eventual delivery of the next generation processor, which will employ the Very High Speed Integrated Circuit (VHSIC) technology. Included within the broad objective is the development of specific applications and demonstration programs for the MRSP.

2. MRSP OVERVIEW

The Multi-mode Radar Signal Processor (MRSP) is configured around the Westinghouse PSP-X Programmable Signal processor. This is a self-contained complete signal processing system capable of operating at very high thru-put rates on real-time data inputs. The MRSP has three major programmable subsystems:

- Vector Arithmetic Processor (VAP)
- Control Arithmetic Processor (CAP)
- Input/Output Controller (IOC).

Additionally, the MRSP has a large block of high speed random access internal storage, which is known simply as the Bulk memory. The VAP, CAP, and IOC each have their own unique assembly language, and each is programmed independently. However, in practical applications, the three programs must work together in a cooperative synergistic manner. This is basically accomplished through an interactive group of flags and interrupts that enable the three programs to communicate with each other.

In the usual signal processing environment, the VAP performs high speed "number crunching" on large arrays of data which have been initially stored in the Bulk Memory. The IOC is responsible for feeding raw data to the VAP in just the right amounts, and at just the right time. In turn, the IOC is responsible for receiving the processed or partially processed data back from the VAP, and returning it to the proper locations in Bulk Memory. Some processing applications, such as a large two-dimensional Fast Fourier Transform (FFT) may require two or more passes through the VAP before the data reaches its final form. Additionally, in a normal environment, the IOC is responsible for feeding the VAP-processed data to the CAP for final disposition. The CAP is the usual MRSP interface to the "outside world". The "outside world" could represent some form of display system if image data were being processed, or it could represent some bulk storage device such as a magnetic tape or disk. In many applications the CAP performs additional processing on the data, such as packing or scaling, before final disposition. The CAP is a powerful general purpose processor in its own right, and has a comprehensive set of instructions. As a system, the MRSP is most efficient when operating with fixed programs on the largest possible data arrays, and in a continuous thru-put manner.

The MRSP is fully capable of operating as an independent system once functioning VAP, CAP, and IOC programs have been loaded, and some method of inputting raw data to the Bulk Memory has been established. However, before it can reach this point of independence, the MRSP needs some "outside" help. A support software package containing such entities as editors, assemblers, linkers, loaders, and debuggers is required to initially generate the programs that are to run in the MRSP. These utilities cannot be independently supported by the MRSP. They must be resident in an external "host" computer. Additionally, there must be a hardware link (interface) between the host and the MRSP so that software generated in the host can be loaded into the MRSP. For this effort, a Digital Equipment Corporation (DEC) VAX 11/750 general purpose computer is used to host the MRSP. The VAX has a very flexible operating system that includes all resources required to fully implement the MRSP support package. Finally, in a practical signal processing

environment, some output device is required to accept the processed data generated by the MRSP. For this application a video display system was selected which consists of a RAMTEK Model 9351 Display Controller, and a CONRAC 17-inch black and white television monitor. This system is capable of supporting a video image size of 512 X 512 picture cells (pixels), with up to 256 different intensity levels per pixel.

The major goal of the MRSP Integration Facility is to gain experience in preparation for the next generation high speed signal processor, which will employ the VHSIC technology. To this end, some special options were included in the MRSP configuration so that it would more nearly resemble the VHSIC brassboard design. For example, the MRSP includes two VAP's which can operate in parallel, thereby reducing processing time by half for certain specific applications. Additionally, the MRSP configuration includes a Differential Address Generator (DAG) option. This is a programmable hardware unit capable of generating non-linear address sequences for the IOC and/or the VAP's. In certain applications, this feature can significantly reduce average processing time by eliminating the need for special address computation prior to each data access. The dual VAP's, along with the IOC, CAP, and DAG option combine to make the MRSP a powerful multi-processor network capable of sophisticated parallel operations. In this regard it emulates the future VHSIC processor, and MRSP operations performed under this effort are expected to ease transition to the next generation processor.

3. PROGRAM MILESTONES

Detailed milestone objectives for the MRSP Integration Facility were established in a hierarchical manner since the actual MRSP hardware was under fabrication, and would not be available for a year after the program was initiated. The first major milestone was the execution of procurement procedures necessary to purchase a host computer and an output display system for the MRSP. This process, requiring approximately five months, included the generation of several economic and feasibility comparisons between 4 candidate computer systems. Many "outside" considerations were involved in the process since this particular computer would support several programs in addition to the

MRSP. A Digital Equipment Corporation (DEC) VAX 11/750 computer system was selected and delivered to AFWAL/AARM in June 1981. Installation and checkout of the VAX required an additional month. A RAMTEK Model 9351 Display Controller and a CONRAC television monitor were selected for the display system, and delivered to AFWAL/AARM in July 1981. These display components were then shipped to Westinghouse in September 1981 for hardware interface with the MRSP.

Once the prospective host computer was in place, the next major milestone objective was installation of the existing MRSP support software package. This was accomplished in August 1981. The support package contains the various assemblers, linkers, loaders, and debuggers necessary to develop actual programs for the MRSP. Even though the actual hardware was not yet available, the support software enabled AFWAL/AARM personnel to gain initial insight into MRSP programming concepts. Additionally, a software package was available from Westinghouse which provided a high-level language (FORTRAN) simulator for the Vector Arithmetic Processor (VAP) element of the MRSP. This is the high speed arithmetic unit which operates on large data arrays, and is the key to the computational power of the MRSP. The VAP simulation package closely simulates each individual VAP instruction with a separate FORTRAN routine. Although much slower than the actual VAP, this package does enable the user to gain experience in programming practical VAP-oriented signal processing problems.

The actual MRSP hardware was delivered to AFWAL/AARM in March 1982. One week of formal training was conducted by contractor representatives in March 1982, and a second week in April 1982. After the MRSP hardware was delivered, the primary objective became one of gaining familiarity and programming proficiency. This newly acquired expertise was then utilized to develop a package of software routines to efficiently integrate the MRSP with the RAMTEK display system. Additionally, several representative programs to demonstrate the processing power of the MRSP were written.

Program development for the MRSP necessarily entails an extensive involvement with the MRSP support software package. This package was delivered with the system, but is not tailored to operate with any specific host computer. The MRSP system user is responsible for providing his own host. The support software design is flexible, and will accommodate a variety of potential host configurations. Consequently, it is probable that some of the support routines can be made to operate more efficiently if certain unique capabilities of a particular host are taken into account. In this regard, an additional objective was established to incorporate VAX-specific refinements into the support software package whenever practical.

SECTION II

MRSP HARDWARE DESCRIPTION

1. OVERALL CONFIGURATION

This section is included for reader continuity. It provides a top level summary of the overall MRSP configuration, and briefly describes the major hardware components. For a more detailed treatment of hardware particulars, the reader is referred to the MRSP system manuals.

Figure 1 is a top level drawing of the basic MRSP system architecture. The MRSP has three major programmable subsystems:

- Vector Arithmetic Processor (VAP).
- Control Arithmetic Processor (CAP).
- Input/Output Controller (IOC)

Additionally, the MRSP has a large block of internal high speed storage (2 megawords X 32 bits) which is known as the Bulk memory. The CAP, VAP, and IOC each has its own unique assembly language, and each is programmed independently. However, in practical applications the three programs must work together in a cooperative manner. Two additional devices, the Differential Address Generator (DAG), and the Direct Memory Access (DMA) Channel Bus Controller may also be considered programmable in a sense. However, these devices do not have unique assembly languages, and are normally loaded from CAP routines.

The MRSP is based on the Westinghouse PSP-X Programmable Signal Processor, which can be fabricated in several possible configurations. This particular configuration incorporates two VAP units which can be operated in parallel for increased computational efficiency. A VAP normally performs high speed arithmetic on large data arrays (vectors) which are complex in nature, i.e., they have a real and an imaginary component. The IOC controls all data transfers into and out of the Bulk Memory. It is a multi-port device, although only one port is active at any particular time. The CAP is an independent general purpose

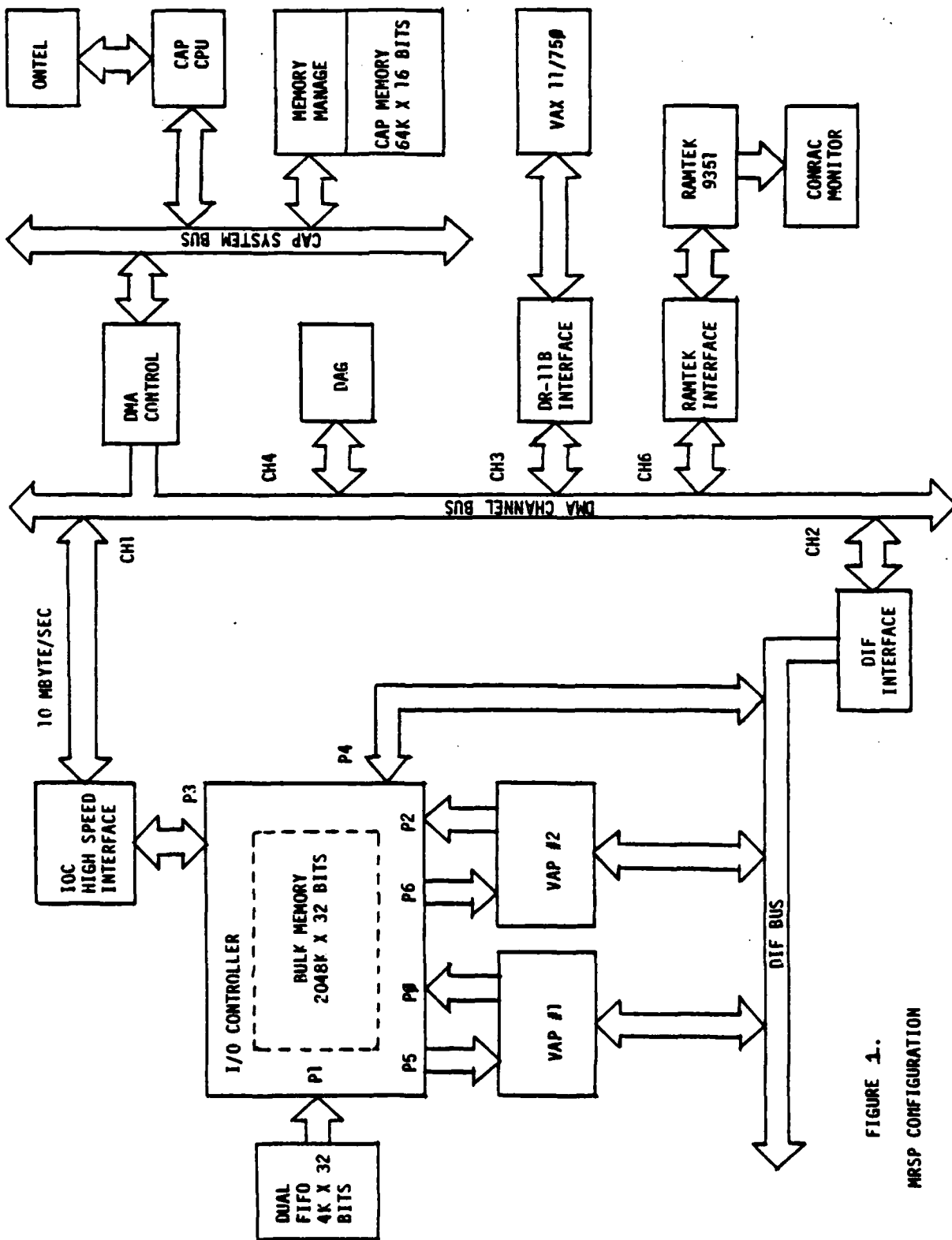


FIGURE 1.
MRSP CONFIGURATION

processor. it communicates with the IOC and all system interfaces through the DMA Channel Bus Controller. The Channel Bus Controller itself has 7 independent data channels, all of which have access to CAP main memory. This particular MRSP configuration also incorporates a Differential Address Generator (DAG) option. The DAG is a separate programmable hardware device capable of directly generating non-consecutive memory address sequences. For certain applications, particularly those involving non-linear interpolations, the DAG can significantly improve processing efficiency by eliminating the need for special address computations prior to each data access from Bulk Memory or VAP data memory. The DAG interfaces to the CAP through a channel on the DMA Bus. A DEC VAX 11/750 computer system serves as host processor for the MRSP. This system runs the MRSP support software package and provides the resources necessary to generate and load programs into the MRSP. The VAX can communicate with the CAP through a channel on the DMA Bus. Additionally, an output video display system consisting of a RAMTEK Display Controller and a CONRAC television monitor is interfaced to the MRSP. This system also communicates with the CAP through a channel on the DMA Bus. Finally, two special options are included with this MRSP configuration. They consist of a First-In-First-Out (FIFO) buffer, and a Console Intelligence Unit (CIU). The FIFO serves to interface the MRSP to a non-synchronous data source, while the CIU is an intelligent CRT terminal capable of a variety of support functions.

In a typical "real time" signal processing environment, the CAP, IOC, and VAP units would operate with fixed programs on a continuous stream of data in "pipeline" fashion. Raw data initially enters the MRSP through the FIFO buffer. This device accumulates data from a non-synchronous source, and passes it to the IOC in blocks for increased efficiency. As an example, the raw data stream could represent unprocessed radar image returns transmitted from an aircraft currently in the air. The IOC would accept data from the FIFO and store it in Bulk Memory. The IOC would then feed the data to the VAP units for some form of complex processing. If the MRSP were operating with radar image data, specialized functions such as demultiplexing, demodulation, frequency filtering, and detection could be performed in the VAP units. After the data has been processed, the IOC transfers it to the CAP.

Assuming that radar image data were involved, the CAP could move the final processed results to an output display system, or alternatively, to some mass storage device such as a magnetic tape or disk. The CAP is a powerful general purpose processor in its own right, and if necessary, could perform some additional data processing operation such as scaling, encoding, or packing before final disposition. If non-linear address sequences were required at any point in the VAP or IOC processing, the DAG could be programmed through the CAP or the host computer with the necessary special functions.

2. SYSTEM COMPONENTS

This section briefly summarizes the major physical and performance characteristics of the MRSP system hardware components.

a. Physical Description

The MRSP is housed in a pair of cabinets, each with a height of 84 inches, a depth of 36 inches, and an outside width of 44 inches. Each cabinet is capable of supporting two separate 50-board chassis, including the required power supplies, monitors, distribution panels, and cooling apparatus. The upper chassis in Cabinet #1 contains the CAP Central Processor Unit (CPU) and memory, the IOC, the Bulk Memory, the DAG, and the DMA Channel Bus Controller. The lower chassis in this same cabinet contains VAP #1. The upper chassis in Cabinet #2 is empty, and the lower chassis contains VAP #2. Integrated circuitry for the MRSP is based on 9 x 12 inch multi-wire boards. Each cabinet contains three (3) five-volt 350-Ampere power supplies, and one (1) two-volt 200-Ampere power supply. Two of the supplies in Cabinet #2 are not actually used, and could serve as spares. Access to the boards and power supplies is provided by three sets of doors on the front of each cabinet, while access to backplane wiring and cabling is provided through two sets of doors at the rear of each cabinet.

b. Vector Arithmetic Processor

The VAP is designed to perform high speed complex operations (real and imaginary components) on large data arrays (vectors). It incorporates a powerful instruction set which includes 3 types of Fast

Fourier Transform (FFT) instructions, 4 different filter instructions, and more than 70 assorted arithmetic, logical, and control instructions. Basic clock for the VAP is 100 nanoseconds, and it is capable of a 10 megahertz "pipeline" processing and thruput rate. The VAP is oriented toward the "butterfly" computations required for FFT operations, and it can perform a full 1024 point-weighted FFT in approximately 0.6 milliseconds. It is also capable of executing a full 16 X 16 bit complex multiply in one clock cycle. From a hardware standpoint, the VAP arithmetic unit includes 4 multipliers and 6 ALU's, which are configured by microprogram control. There are 6 data memories, each 4096 words X 32 bits. The VAP also has dual input and dual output buffers. Each input/output buffer memory is 4096 words X 32 bits. The microprogrammed control structure implemented in the VAP enables an experienced user to customize his own specialized instructions for increased efficiency.

c. Control Arithmetic Processor

The CAP is a 16-bit architecture general purpose processor, but is capable of 32-bit double precision and floating point operations. It has a powerful instruction set consisting of approximately 200 assorted arithmetic, logical, and control instructions. Several addressing modes are possible, including direct, indirect, indexed, and immediate. High level languages, including JOVIAL and FORTRAN are supported by the CAP. The central processing unit includes 16 general purpose registers, 32 vectored interrupts, and a memory management capability. The CAP has a 64K X 16-bit main memory, which could be expanded up to one megaword with additional memory boards. By means of the DMA Channel Bus Controller, the CAP can access up to 7 external/internal interfaces, at rates of up to 8 megabytes/second.

d. Input/Output Controller and Bulk Memory

The IOC is a microprogrammed processor which formats and transfers data between the Bulk Memory and the VAP, CAP, host computer, and external high speed buffers. It is capable of 100 nanosecond sequential read/write operations. The IOC has 8 independent ports, each accessing Bulk Memory, and each controlled by its own set of instructions in the program memory. It has a port priority control system, and 32

flags which can be set or sensed by port programs to provide synchronization between subsystems. The IOC instruction set provides complete flexibility in moving data into and out of the Bulk Memory. Addresses can be updated in either an indexed or an offset mode, or non-linearly generated VIA the DAG. Large blocks of data can be moved with a single instruction. Any desired packing factor, i.e., full 32 bits, upper 16 bits, or lower 16 bits can be incorporated into any instruction. The Bulk Memory itself consists of 2 megawords X 32 bits of high speed random access storage.

e. Direct Memory Access Channel Bus Controller

The DMA Channel Bus Controller provides a programmable interface between any of the external devices connected to the Channel Bus, and the main memory of the CAP processor. It also serves as a direct interface between any two of the devices. The DMA Controller is capable of managing up to 7 independent channels. At any particular instant only one channel has control of the bus, but at that instant there could be up to 7 transfers in various stages of completion. This "interleaved" operation enhances traffic flow efficiency on the bus. Details pertinent to Channel Bus transfers are specified by up to four 16-bit control words. Each channel has its own control word sequence, and each has its own "hard" registers for control word storage. The DMA Channel Bus Control words may be loaded in either of two ways. First, any control word sequence can always be loaded directly by the CAP through the use of hard register instructions. Additionally, it is possible for some interfaces to load their own control words. In this latter case, the first words received by the channel would be interpreted as the control words. These would then specify how the following words were to be handled.

f. FIFO Interface

The MRSP configuration includes an optional First-In-First-Out (FIFO) buffer which bypasses CAP or host control to move data at high Speed directly into Bulk Memory. This is actually a double buffered device designed to accept data at the input source rate, and transfer it at the output destination rate. In effect, it matches the MRSP to a

non-synchronous data source. The FIFO communicates directly with an assigned input port in the IOC. Most FIFO operations are controlled by a programmable read only memory (PROM), which is set to the users specifications. However, some parameters, such as mode of operation and block size can be loaded under program control. The FIFO memory consists of two identical buffers, each 4096 words X 32 bits. While one memory is being written, the other may be read (unloaded). This allows a continuous input data stream and simultaneous transfer to the IOC. Each 32-bit wide buffer is divided into four 8-bit bytes, and the user may pack the incoming 8-bit bytes in any order desired.

g. Differential Address Generator

The MRSP configuration includes optional DAG hardware which is capable of generating non-consecutive address sequences for Bulk Memory and/or VAP data memory. This feature can potentially increase processing efficiency for certain types of interpolation operations. In effect, the desired address sequences would be generated directly by hardware rather than computed by the program. The DAG can produce address sequences in which the resampled vector is either linear or quadratic relative to the initial reference. DAG outputs are sent to an Address Offset Table (AOT) located in the IOC and/or to either or both of two Pointer Tables (PT1 and PT2) located in the VAP. DAG operations are controlled from a register file containing 32 words X 16 bits, which is loaded by either the CAP or a host computer. Values from this file are then used to initialize various sum, increment, offset, carry, and compare registers within the DAG hardware. The derivation of DAG register file values can be an involved process requiring much planning on the part of the user, especially for second order sequences. The DAG is assigned to Channel #4 on the DMA Bus.

h. Console Intelligence Unit

The MRSP configuration includes a supplemental ONTEL OP-1 Intelligent Terminal, which serves as a Console Intelligence Unit. This is a microprocessor based (INTEL 8080) unit which provides standard computer console functions through a CRT/keyboard. A dual drive floppy disk peripheral is included with this unit. The ONTEL can function as a

completely independent system, and is also capable of full interaction with the CAP processor. It has its own floppy based operating system which is called DOS/80. The ONTEL CIU is most useful for dynamic debugging of CAP programs. It can monitor and modify CAP registers and memory, and can execute CAP programs in single step fashion. The ONTEL can also save CAP programs on floppy disk. Additionally, it can load CAP programs from the disk and execute them. It is also possible to use the ONTEL CRT as a supplemental alphanumeric output display for normal MRSP operations. The ONTEL is tied to the CAP through a unique CIU interface.

i. RAMTEK Display System

The RAMTEK video display system serves as the primary output device in the present MRSP configuration. It consists of a RAMTEK Model 9351 Display Controller, a CONRAC 17-inch black and white television monitor, and a manual joystick cursor controller. The system has sufficient refresh memory to support an image size of up to 512 X 512 picture cells (pixels), with each pixel being represented by up to 8 bits of intensity data (256 grey levels). Any pixel in refresh memory may be randomly accessed. The RAMTEK Controller has a comprehensive instruction set which enables it to plot functions, generate text and graphics symbols, and display image intensity data in a raster format. The capability to load variable look-up tables, reverse display polarity, blink the display, and generate a joystick controllable cursor is also included. To control the system, unique RAMTEK instruction codes are programmed as data words within a CAP assembly language routine. The RAMTEK video display system communicates with the MRSP through Channel #6 on the DMA Bus, and CAP Interrupt #6.

j. VAX 11/750 Host Computer

A Digital Equipment Corporation VAX 11/750 computer system serves as a host processor for the MRSP. The VAX is a 32-bit medium-scale general purpose machine. It is not dedicated to the MRSP, and supports several additional programs. The VAX has an extremely

AFWAL-TR-83-1045

comprehensive and flexible operating system (VAX/VMS), which is based on virtual memory management techniques. In its role as host, the VAX executes the MRSP support software package, and provides the resources required to generate and load runnable binary CAP, VAP, and IOC modules. The VAX is hardware linked to the MRSP through an MDB Corp. plug-compatible equivalent of a DEC DR-11B interface. It communicates with the CAP through Channel #3 on the DMA Bus.

SECTION III

MRSP SYSTEM SOFTWARE DESCRIPTION

1. GENERAL OVERVIEW

This section is included for reader continuity. It provides a top level summary of the MRSP Support Software Package, which is the primary user/software interface required for normal MRSP system operations. A separate Westinghouse software package which provides a high level language (FORTRAN) simulator for the VAP element of the MRSP is also described. Finally, some directly applicable features of the host processor operating system (VAX/VMS) are briefly highlighted. For a more detailed description of software particulars, the reader is referred to the MRSP and VAX system manuals.

2. SUPPORT SOFTWARE OVERVIEW

The individual routines required to assemble, link, load, debug, execute, and modify useful programs for the MRSP comprise an overall system of modules called the MRSP Support Software Package. This is a general purpose package written in a machine-independent higher level language. It cannot be executed or run directly by the MRSP hardware, but must reside in a separate host processor which supports that particular language. A DEC VAX 11/750 system serves as host for this particular MRSP configuration. The MRSP Support Package does not include a text editor, which is required to generate initial source code programs for the VAP, IOC, and CAP. An editor must be provided by the VAX host. Additionally, some VAX resources are required to create the final binary coded load module.

Generation of a runnable program for the VAP, IOC, or CAP is a multi-step process. It is accomplished from a standard keyboard/CRT computer terminal supported by the VAX host processor. First, the user invokes one of the VAX text editors, and types the source program in the assembly language for that particular unit. The VAP, IOC, and CAP each have their own assembly language mnemonics. After the source program has been written, the appropriate assembler is invoked. The assembler

creates a file in which the mnemonics and symbols used in the source program are converted into numeric machine instruction codes and addresses. Next, the assembler output file is operated on by a VAX-specific command file called "BINGEN" (Binary Generator). BINGEN creates still another file in which the assembler code is converted into a loadable binary format. Finally, the loader itself is invoked, which moves the runnable binary program module into the MRSP.

The BINGEN command file operates on assembled programs in a particular ASCII-octal format. This format is generated directly by the VAP and IOC assemblers. However, the output from the CAP assembler is in an absolute binary format which BINGEN does not recognize. An additional step is required to convert assembled CAP program files into the desired format. The utility which performs this conversion is called the CAP Translator. It is invoked prior to the BINGEN operation. All three assemblers have the option of generating an additional list file, which is printed in human-intelligible ASCII characters. This file contains the source mnemonics and symbols in tabular form. An adjacent column shows the equivalent machine operation codes and translated symbolic addresses. An additional adjacent column indicates the memory location in which each machine coded instruction and operand will be stored when the program is eventually loaded. The list file is particularly useful for program debugging purposes.

BINGEN is an interactive VAX-specific command file which uses VAX resources to convert a file in ASCII-octal format into a loadable binary image file. BINGEN prompts the user for the name of the unit to be loaded (VAP, CAP, or IOC), and then for some specific parameter definitions. The nominal values for these parameters are stored in a separate default file. If the user desires the default values, he simply types a carriage return in response to the prompt. BINGEN then requests the name of the ASCII-octal format file which is to be converted. For the actual binary conversion process, BINGEN invokes the VAX Macro assembler and the VAX Compatibility Mode task-builder. Finally, after the required processing has been accomplished, BINGEN asks for the name of the loadable file to be generated.

Once an operational binary program file has been created in the host processor, the loader segment of the MRSP Support Software Package is invoked. This segment provides the user with a method to physically load a runnable program into the MRSP. In addition to actually loading programs, the loader has the ability to make specific load-time changes to a module in accordance with a previously created edit file.

When a new operational program is first tested in the VAP, CAP, or IOC, the probability that some subtle programming errors or "bugs" will exist is quite high, especially if sophisticated data manipulations are involved. For this reason, a Debugger module is available in the MRSP Support package which enables the user to locate and correct programming errors. The Debugger permits direct interactive access with the VAP, CAP, IOC, or DAG. The user can examine and modify the contents of data memories and internal register memories within the specified MRSP unit. Additionally, he can set breakpoints, execute the program on a complete or single step basis, and reset the processor. These operations are all performed from a standard keyboard/CRT which is supported by the host processor. In addition to its main function of debugging new programs, the MRSP Debugger module has a second important purpose. It provides a means to execute any program which has been loaded into the VAP, CAP, or IOC. For this application, the Debugger is normally invoked immediately after the loader. The operator then selects the desired MRSP unit and sends an EXECUTE command.

3. CAP SUPPORT PACKAGE

The CAP Support Package is a segment of the overall MRSP support package. It provides a number of supplemental service routines unique to programs written for the CAP. These routines include a File Manager, a Linker, a Simulator, and a Deassembler.

The CAP File Manager is a general utility which has been designed to maintain and keep track of CAP source and relocatable (binary-coded) program files. It is useful for such operations as adding new code segments to existing files, deleting files, copying files, or renaming files.

The CAP Linker is a routine which collects independently assembled relocatable program elements, and binds them into a single load module. Undefined symbols within a particular relocatable element are resolved through reference to externally defined symbols in the other relocatable elements. This effectively enables communication between the independently assembled elements when the program is executing.

The CAP Simulator precisely duplicates the operational behavior of the CAP Processor, and provides a variety of detailed outputs. In operation, the simulator maintains a software replica of CAP memory within the host processor. The program to be simulated is assembled in the normal manner, just as it would be with the actual CAP. The output of the simulator is a detailed sequential listing of all CAP instructions that were executed, and a dump of all programmable registers. The printout also includes the actual execution time of each instruction, as well as the total accumulated program execution time to that point.

As one of the assembler options, the executable binary module can be deassembled, producing as well as possible, an assembly language source listing of that program. However, this can only be done at assembly time.

4. HOST I/O DRIVER

The Host Input/Output Driver is one of the most useful entities in the MRSP Support Package. It provides a software communications link between the host processor and the MRSP. The Host I/O Driver consists of a number of separate modules which can perform such functions as the following:

- a. Assign a host I/O channel to the MRSP.
- b. Transmit data back and forth between a high level language applications program running in the host and the MRSP.
- c. Print I/O status code messages.
- d. Load CAP memory management registers.
- e. Reset a specified MRSP unit, or execute a program in that unit.

The major user interface module within the Host I/O Driver package is called PSPIO. This module may be linked to a high level language applications program running in the host, and called as a subroutine. When used as a subroutine, the PSPIO module requires 8 parameters. These are detailed as follows:

- a. Function - the actual operation that is to be performed on this transfer, i.e., read, write, execute, or initialize.
- b. Logical Unit Number - the assigned host processor I/O channel for this transfer.
- c. Unit Descriptor - the name of a file that contains parameters to identify the particular MRSP unit with which the transfer is to occur, i.e., VAP, CAP, IOC, Bulk Memory, or DAG.
- d. Starting Address - the absolute memory address within the object MRSP unit at which the transfer is to begin.
- e. Word Count - the actual number of words to be sent or received on this transfer.
- f. Buffer - the name of a memory array in the host processor from which data is to be sent or received on this transfer.
- g. Time Out Count - A parameter which determines how long the host processor will wait for a response once a transfer has been initiated. If no response is received within this interval, an error message will be generated.
- h. Status - a one-word code indicating either that the transfer was completed normally or that an error occurred. If the transfer is not completed normally, the code identifies the particular error.

PSPIO is extremely useful in situations where MRSP processing is required to support a high level language applications program running in the host processor. It effectively places the full computational power of the MRSP at the disposal of the applications program on an interactive basis. Some specific examples in this regard could include the transmission of special function data directly to the DAG register file, the transmission of weighting functions to VAP data memory for FFT

operations, or the transmission of interactive user inputs directly to CAP memory for MRSP output interface processing.

5. VAP SIMULATOR

The VAP Simulator is a Westinghouse-developed package which provides a high level language (FORTRAN) simulation of the Vector Arithmetic Processor (VAP) element of the MRSP. It may be used to support the development, analysis, and evaluation of VAP-specific algorithms. Use of the simulation package requires knowledge of the MRSP and its functions, particularly the interaction between the IOC and the VAP. Specific knowledge of the VAP instruction set, and the meaning of each instruction field is also required.

The VAP simulation package contains a set of FORTRAN subroutines which simulate the most commonly used VAP instructions. This would include the Fast Fourier Transform (FFT) instructions, the DETECT instruction, and a repertoire of arithmetic instructions that perform operations with complex numbers. Some logical and control instructions are also included. Each simulated instruction is represented by a unique subroutine which has the same name as the actual VAP instruction. Control fields in the actual instruction are represented by subroutine parameters in the simulated instruction. VAP memory is simulated by a large COMMON storage block, which the calling FORTRAN program reserves in the host processor memory. In the simulation package, the large COMMON block is partitioned into 10 sub-blocks of 2048 integers. These sub-blocks are further partitioned into pairs, each representing a complex number (real and imaginary components). The 10 sub-blocks represent the 6 VAP data memories plus the two input buffers and the two output buffers. Subroutines in the VAP Simulator package communicate with one another through reference to the COMMON block.

To use the VAP Simulator, the analyst develops a main FORTRAN calling program which reserves the COMMON block and defines the algorithm that is to be implemented. Each VAP instruction in the algorithm would be represented by a call to the appropriate subroutine. Parameters passed to the subroutine would contain the same information as

the corresponding fields in the actual VAP instruction. The simulated instructions are executed in exactly the same order as real instructions in an actual VAP program. Since it is executing entirely within the host processor, the VAP simulator is, of course, much slower than the actual VAP processor. However, an algorithm implemented by the VAP simulator should yield exactly the same results as it would in the actual VAP. For this reason, the simulator is extremely useful for the initial development of the VAP-oriented signal processing algorithms.

6. SYSTEM DIAGNOSTICS

The MRSP Support Package includes an extensive repertoire of hardware diagnostic routines which verify proper operation of the MRSP/host processor interface, as well as each individual hardware module within the MRSP itself. The diagnostic package is organized into a hierarchical structure, such that the highest level tests the MRSP/host interface, the next highest level tests the major MRSP hardware units (i.e., CAP, VAP, and IOC), and successively lower levels test individual components within each major unit. At the lowest levels, each individual instruction within the VAP, CAP, and IOC is exercised; and detailed patterns are generated to test each cell in the main data memories and internal register memories. Any individual diagnostic routine at any level may be executed as an independent module for selective test of a particular hardware component. The diagnostics package is executed from a standard keyboard/CRT terminal supported by the host processor.

7. HOST PROCESSOR OPERATING SYSTEM

From a practical standpoint, all user interactions with the MRSP must necessarily include the host processor and its operating system. As previously indicated, a VAX 11/750 serves as the primary host processor to the MRSP for this effort. The VAX operating system (VAX/VMS) is extremely comprehensive, and its interactions with the MRSP Support Software package are quite complex. For example, VAX/VMS supports the computer terminals that MRSP system users employ to develop, load, and debug programs. It provides text editors which are required for MRSP source program development, and it supplies resources required for the previously described BINGEN operation. Modules in the Host I/O Driver

package depend on the invocation of various VAX System Service routines to fulfill their function. One of the most useful features of VAX/VMS is the Digital Command Language (DCL). This is a unique console language that functions at the operating system level. DCL provides the ability to generate command procedures, which in effect, are files of commands at the operating system level. These files enable the VAX to automatically execute a long string of system level commands that would ordinarily require individual operator input. This capability can greatly streamline MRSP system operations from a user standpoint. For example, it will be recalled that the procedure to transform a CAP program from ASCII source code to a loaded binary module involves successive invocations of the CAP Assembler, CAP Translator, BINGEN, MRSP Loader, and MRSP Debugger. Each of these invocations entails a number of operating system level commands on the part of the user. The entire process could be simplified by creating a DCL command procedure that contains the required string of commands, including any parameters that would be entered manually. Then the user need only specify the name of the command procedure, and the whole sequence is executed automatically. Command procedure files are especially helpful in situations where the MRSP interacts with a number of high level language applications programs running in the VAX. As an additional example, suppose that it is desired to do some FFT processing on a large array of data, and then move the results to a display system controlled by the CAP. To accomplish this, one could first issue commands to load the operational VAP, CAP, and IOC programs into the MRSP. This would be followed by a command to execute a FORTRAN program that reads a magnetic tape and moves the data to Bulk Memory. Another FORTRAN program would then be executed which generates an FFT weighting function and moves it to VAP data memory. Finally, commands would be issued to execute the individual VAP, CAP, and IOC programs. If it were desired to also save the processed data on magnetic tape, then an additional FORTRAN program would be executed to accomplish this function. All of these commands could be placed in a single DCL file, and automatically executed in sequence by simply specifying the name of that file. Any parameters that would ordinarily require manual entry can also be included in the command file.

SECTION IV

MRSP SYSTEM OPERATIONS

1. RAMTEK DISPLAY INTEGRATION

The video display system interfaced to the MRSP consists of a RAMTEK Model 9351 Display Controller, and a CONRAC 17-inch black and white television monitor. The RAMTEK controller includes a 512 X 512 X 8 bit digital refresh memory and a 2048 X 13 bit video look-up table. The refresh memory permits the system to display up to 262, 144 individual picture cells (pixels) on the monitor screen. Each pixel can be coded to any of 256 different grey levels. The look-up table determines how each individual pixel will be coded. Any desired mapping function (linear, logarithmic, exponential, etc.) may be loaded into the look-up table under program control. Since the refresh memory in this particular system is only 8 bits deep, only 256 of the available 2048 look-up table locations would be used in practical applications. The RAMTEK unit includes hardware to control an internally generated cursor symbol, either under program control or manually by means of a joystick. Additionally, the RAMTEK Controller includes circuitry to generate alphanumeric characters and graphics (vectors). The CONRAC monitor has been tuned and adjusted to match the RAMTEK controller. In most references within this report, the MRSP display system will be conveniently referred to as the "RAMTEK display" or simply as "the RAMTEK". However, these references implicitly include the CONRAC monitor.

The RAMTEK display system is interfaced to the MRSP through Channel #6 on the DMA Channel Bus Controller. It operates through a CAP program which controls all the required interrupt processing and "handshake" operations. Under normal implementation, the CAP program performs some processing operation on the data, and then forwards it to the RAMTEK for display. Every CAP program which uses the RAMTEK must incorporate a series of RAMTEK instruction words. These instructions are programmed as data words in the CAP program, while the actual data to be displayed is represented as a CAP memory array.

The RAMTEK controller provides a repertoire of 20 basic instructions which can be used to manipulate data presented on a CRT display. There are 16 possible parameters through which the programmer can "customize" instructions executed by the RAMTEK. Not all parameters apply to all instructions, and some of the parameters are not used in the Model 9351. However, the system is designed for optimum flexibility, and the number of possible instruction/parameter combinations is very large. As would be expected, the usual price for flexibility of operation is complexity of programming. Any RAMTEK operation requires a series of 16-bit instruction words which precede the actual data to be displayed. These words effectively constitute a unique RAMTEK "subprogram" within the CAP program. The first word includes an 8-bit op-code field and several smaller fields which specify such factors as addressing mode, background polarity, byte packing, and whether additive write is to be used. This is followed by an operand flag word which indicates the presence or absence of the 16 possible parameter operands. The operands appear in a fixed sequence, and each bit in the flag word corresponds to an operand in the same sequence. The actual number of words associated with each parameter varies from 1 to 12, depending on the parameter. These words directly follow the flag word. The operand words are followed by a byte count word, and finally by the data words themselves. The RAMTEK instruction set and parameter formats are summarized in Figure 2 and 3.

From the above it is apparent that the programmer is literally working at the bit level, and constantly referring to detailed manuals to derive the various op-codes, control fields, flag words, operands, and byte counts. Moreover, this process must be repeated for each CAP program that communicates with the RAMTEK display. The RAMTEK display integration effort can be summarized as an attempt to move some of the more useful RAMTEK instructions to a higher level from which they could be directly referenced by descriptive names. This would minimize the need for bit level coding in each individual program, which is a tedious process and highly prone to error.

OP- CODE	INSTRUCTION NAME
01	LOAD HARD REGISTER
02	READ SOFT REGISTER
03	LOAD AUXILIARY MEMORY
04	READ AUXILIARY MEMORY
05	RESET
06	INITIALIZE
07	NO OPERATIONS
08	SET PARAMETER
09	ERASE
0A	WRITE IMAGE
0B	READ IMAGE
0C	WRITE TEXT
0D	WRITE RASTER
0E	WRITE VECTOR
10	WRITE PLOT
16	WRITE CURSOR STATE
17	READ CURSOR STATE
18	WRITE KEYBOARD
19	READ KEYBOARD
1A	SENSE PERIPHERAL STATUS

Figure 2. Basic Ramtek Instruction Set

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

OPERAND BIT	ARGUMENT
0	SUBCHANNELS
1	FOREGROUND
2	BACKGROUND
3	INDEX 1
4	INDEX 2
5	ORIGIN
6	WINDOW
7	SCAN
8	DIMENSION
9	SPACING
10	SCALE
11	FUNCTION
12	CONIC EQUATION
13	BASELINE
14	SCROLL COUNT
15	START POINT

OP CODE	IX	AD	BK	RP	OF	DF
OPERAND FLAG						
SUBCHANNELS MASK						
FOREGROUND COLOR						
BACKGROUND COLOR						
X ADDRESS						
Y ADDRESS						
X ADDRESS						
Y ADDRESS						
X ADDRESS (NOT USED)						
Y ADDRESS						
START X ADDRESS						
START Y ADDRESS						
STOP X ADDRESS						
STOP Y ADDRESS						
SCAN SEQUENCE						
FONT/SEGMENT WIDTH						
FONT HEIGHT						
HORIZONTAL SPACING						
VERTICAL SPACING						
Y SCALE X SCALE						
NOT USED						
NOT USED						
NOT USED						
NOT USED						
NOT USED						
NOT USED						
FILLED PLOT BASELINE						
NOT USED						
X ADDRESS						
Y ADDRESS						
DATA BYTE COUNT						
DATA						

LEGEND

IX ADDRESSING MODE (0 ABSOLUTE 1 INDEX 1 2 INDEX 2 3 RELATIVE)
 AD ADDITIVE WRITE (0 REPLACEMENT 1 ADDITIVE)
 BK REVERSE BACKGROUND (0 NORMAL BACKGROUND 1 REVERSED BACKGROUND)
 RP REVERSE PACKING FLAG (0 LEFT BYTE FIRST 1 RIGHT BYTE FIRST)
 OF OPERAND FLAG (0 NO ARGUMENT OR FLAG WORKS EXISTS 1 FLAGGED ARGUMENTS EXIST)
 DF DATA FLAG (0 NO DATA OR LENGTH WORD EXISTS 1 DATA BYTES)

Figure 3. Instruction Parameter Format

a. General Approach

In the normal MRSP software development environment, the most frequently used RAMTEK instructions are those which write image, text, and graphics data to the CRT display. Instructions which clear the display, control the cursor, and load the auxiliary memory with a lookup table are also very useful. One of the major technical objectives of this effort was to integrate the RAMTEK display from a software standpoint, and make it easier to use from an operator standpoint. This was accomplished primarily through the features of the VAX VMS Operating System, and the PSPIO Utility Routines in the MRSP support package. The VAX Operating System allows the programmer to create command procedure files. Such files enable the VAX to execute a long series of commands at the operating system level that would ordinarily require individual operator input. Once a command file has been created, the operator need only specify the name of the file and the VAX does the rest. The PSPIO utility also has some handy features. One of the most useful is to allow a VAX FORTRAN program to directly and independently send data to any desired location in CAP memory. It also enables a VAX FORTRAN program to command the execution of an assembly language program loaded into VAP, CAP, or IOC.

The approach used in integrating the RAMTEK software was to write separate VAX command files for individually implementing the most useful RAMTEK instructions. For example, VAX command files named WVECTOR and WTEXT were written to implement the basic RAMTEK WRITE VECTOR and WRITE TEXT instructions respectively. Some of the command files are interactive in that they dynamically request operator inputs directly from the keyboard terminal at run time. Others require previously created data files, while still others execute directly without operator inputs or data files. All of these command files have one essential feature. They convert the bit level RAMTEK operand codes to higher level mnemonics which can be entered directly from the keyboard of an ordinary computer terminal.

Each VAX command file representing a RAMTEK instruction is designed to operate as an independent module. In general, these files perform five basic functions, which are summarized below:

- (1) Accept interactive operator parameter inputs directly from the keyboard terminal and store in a temporary data file.
- (2) Load a CAP assembly language program which incorporates the bit level codes of the desired RAMTEK instruction, and also reserves storage for parameter data.
- (3) Run a VAX FORTRAN program (incorporating the PSPIO utility) which processes the interactive parameter data and sends it directly to the specially reserved locations in CAP memory.
- (4) Execute the CAP program which now has all required parameter data. This is actually accomplished by a second FORTRAN program incorporating the PSPIO utility.
- (5) Perform "housekeeping" functions, including the deletion of all useless temporary and intermediate data files.

b. Command File Descriptions

The following VAX command files were written during the course of this effort:

- (1) WTEXT - this file implements the RAMTEK WRITE TEXT instruction to place alpha-numeric characters at an operator selectable starting point on the CRT display.
- (2) WVECTOR - this file implements the RAMTEK WRITE VECTOR instruction to draw vectors at operator selected endpoints on the CRT display.
- (3) WPLOT - this file implements the RAMTEK WRITE PLOT instruction. It scales and plots any 512 point function on the CRT display.
- (4) WLOOKUP - this file implements the RAMTEK LOAD AUXILIARY MEMORY instruction. It loads any 256 point lookup table into the RAMTEK Display Controller.

(5) WIMAGEU - this file implements the RAMTEK WRITE IMAGE instruction. It scales and displays a 512 X 512 point image from the upper 16 bits of MRSP bulk memory.

(6) WIMAGEL - this file implements the RAMTEK WRITE IMAGE instruction. It scales and displays a 512 X 512 point image from the lower 16 bits of MRSP bulk memory.

(7) WCURSOR - this file implements the RAMTEK READ CURSOR, WRITE CURSOR, and SENSE PERIPHERAL instructions. It places a crosshair cursor on the CRT display which is manually controllable from a joystick. Cursor coordinates are continuously updated and presented on the supplemental ONTEL display.

(8) RAMCLEAR - this file sends a MASTER CLEAR instruction to the RAMTEK Controller and completely erases the CRT display. The lookup table is not affected.

(9) NEGATE - this file loads a negative lookup table into the RAMTEK Controller, and changes the polarity of any vector and text data currently displayed.

(a) WTEXT

The RAMTEK WRITE TEXT instruction reads ASCII character codes from the host and generates the corresponding text characters for display. Each character is written into a rectangular patch of 9 pixels high by 7 pixels wide. Figure 4 shows the available characters and the font size. Use of this instruction requires that ASCII characters be sequentially packed in 8-bit bytes. Each 16-bit data word sent to the RAMTEK would thus contain two characters. These data words are preceded by a 16-bit flag word containing the byte count, i.e., the number of characters in the message to be displayed. The START POINT parameter is used to specify the starting point of the message.

To make the WRITE TEXT instruction easier to use, the WTEXT command file was written. This file enables the user to input text data directly from the keyboard of his terminal, or at his option, to display an alpha-numeric message directly from a previously created

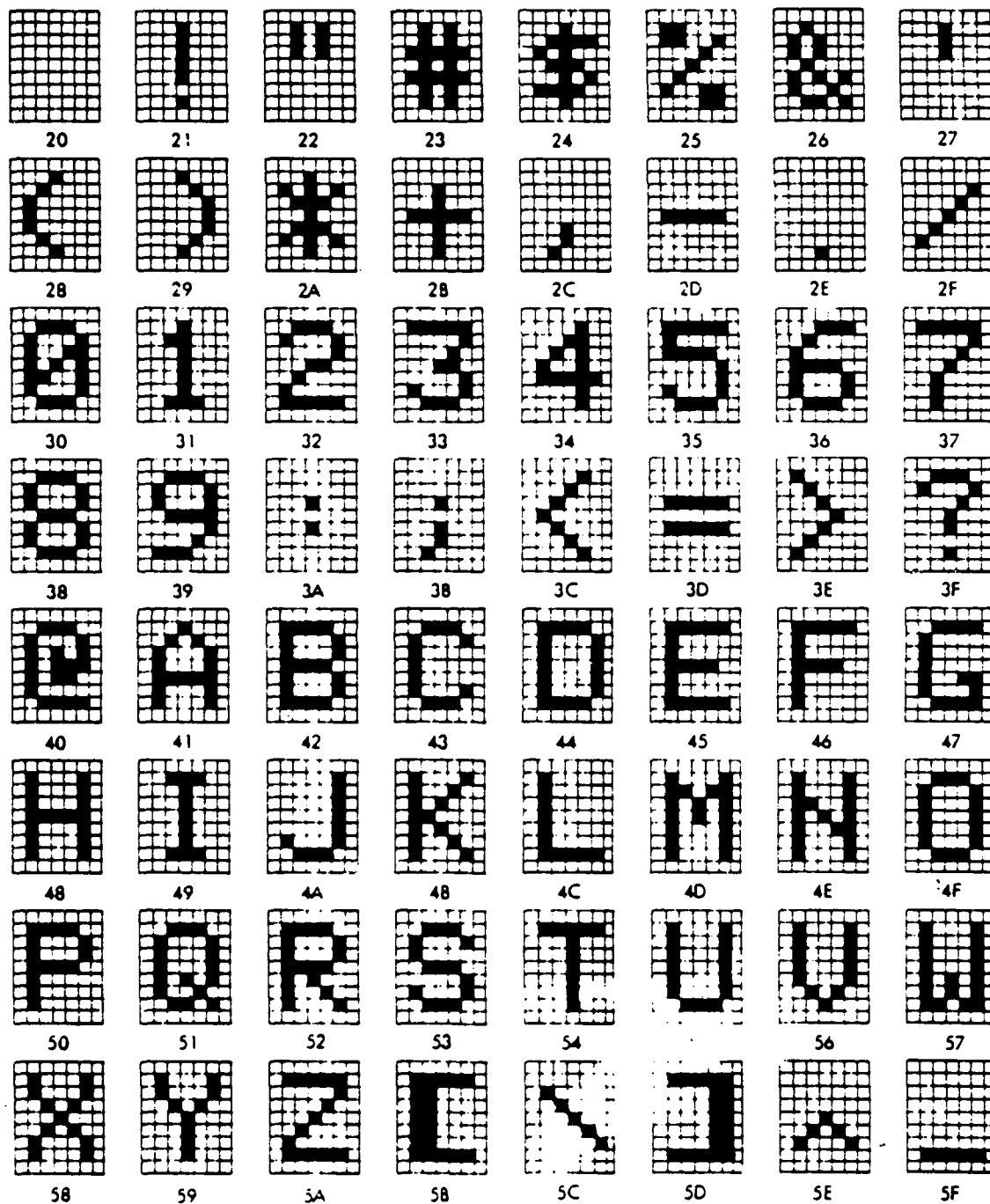


Figure 4. Standard Text Character Fonts

data file. To enter a message from the keyboard the user simply types @ WTEST. Assuming that the user is working from a standard CRT computer terminal, a prompt appears on the screen requesting the X and Y starting coordinates of the alpha-numeric message. These are entered by the user, each followed by a carriage return. A prompt then appears requesting a message directly from the terminal keyboard. The user enters the desired characters and concludes with a carriage return. At this point the RAMTEK, under control of the CAP, displays the message at the desired location on the CRT screen. A prompt also appears on the user's terminal requesting a "YES" or "NO" answer to the question of whether the user wants to save the message in a data file. If "YES" is selected, a follow-up prompt requests the name of the data file to be created; and the internal file containing the message is automatically given the desired name. If the user does not wish to save the text he types a "NO" in response to the prompt, and the internal text file is automatically deleted. If the user wishes to display a message from a previously created data file he enters @ WTEXT FILE-NAME, where "FILE-NAME" is the name of data file containing text and location data. No terminal prompts are generated in this case. The message from the file is sent directly to the RAMTEK and displayed on the CRT screen.

The WTEXT command file uses the VAX Digital Command Language (DCL) to generate the interactive inputs from the operator. These are stored in an intermediate data file. A VAX FORTRAN program is then executed which reads the intermediate file, processes the text data, and sends it to CAP memory VIA the PSPIO utility. Next a CAP assembly language program which moves the data to the RAMTEK is loaded. This CAP program is executed by a VAX FORTRAN routine utilizing PSPIO. Finally, the last task of WTEXT is to dispose of the intermediate data file. If "YES" was selected in response to the save question, the intermediate file is given the name selected by the user, otherwise it is automatically deleted. If WTEXT was operating on a previously created data file, that file is preserved.

(b) WVECTOR

The RAMTEK WRITE VECTOR instruction draws a continuous straight line vector between a starting point and a specified end-point. If more than one vector end-point is specified, then contiguous straight lines will be drawn between the points. It is not possible to draw more than one vector with the same starting point in the same WRITE VECTOR instruction. Multiple instructions must be used if more than one vector is to proceed from the same point. However, it is possible to draw a continuous closed figure which returns to the original starting point with only one instruction. It is also possible to retrace all or part of a previously drawn vector within the same instruction. Therefore, in practice, multiple instructions would only be required in cases where there is a discontinuity (open space) between the vectors to be drawn. Use of this instruction requires the X and Y coordinate of an end-point to be stored in separate 16-bit data words. Thus each end-point to be specified requires two 16-bit words. These data words are preceded by a 16-bit flag word containing the byte count, i.e., the total number of 8-bit bytes of data to be sent to RAMTEK. The byte count is 4 times the number of end-points specified. The start point may be either the Current Operating Point (COP) from the previous instruction, or a new starting point specified by the START POINT parameter.

To make the WRITE VECTOR instruction easier to use, an interactive command file was written for the VAX called WVECTOR. This file enables the user to input vector end-points directly from the keyboard of his terminal, or at his option, to draw vectors from end-points stored in a previously created data file. To enter vector end-points from the keyboard, the user simply types @ WVECTOR. Assuming that the user is working from a standard CRT computer terminal, a prompt appears on the screen requesting the X and Y coordinates of vector end-points. The X and Y coordinates of each end-point to be specified are entered in pairs, separated by a comma and terminated with a carriage return. The prompt reappears after each pair of coordinates has been entered. When all desired end-points have been entered, the user types "END" in response to the prompt. The RAMTEK, under control of the CAP, then draws the specified vectors on the CRT screen. At the same time a

prompt appears on the user's terminal requesting a "YES" or "NO" answer to the question of whether the user wants to save the end-points he has just entered in a data file. If "YES" is selected, a follow-up prompt requests the name of the data file to be created; and the internal file containing the endpoints is given the desired name. If the user does not wish to save the data he types "NO" in response to the prompt, and the internal file is automatically deleted. If the user wishes to draw vectors from a previously created data file he enters @ WVECTOR FILENAME where "FILENAME" is the name of data file containing vector end-points. No terminal prompts are generated in this case. The end-point data from the file is sent directly to the RAMTEK and displayed on the CRT screen.

The WVECTOR command file uses the VAX Digital Command Language to generate the interactive inputs from the operator. These are stored in an intermediate data file. A VAX FORTRAN program is then executed which reads the intermediate file, processes the vector data, and sends it to CAP memory VIA the PSPIO utility. Next a CAP assembly language program which moves the data to the RAMTEK is loaded. This CAP program is executed by a VAX FORTRAN routine using PSPIO. Finally the last task of WVECTOR is to dispose of the intermediate data file. If "YES" was selected in response to the save question, the intermediate file is given the name selected by the user, otherwise it is automatically deleted. If WVECTOR was operating on a previously created data file, that file is preserved.

(c) WPLOT

The RAMTEK WRITE PLOT instruction generates plot segments for each 16-bit word present in the instruction data stream, and automatically updates the current operating point after each segment is generated. This instruction is very flexible, and considerable latitude is available to the programmer as to the orientation, size, height, and spacing of the generated plot. There is also a baseline parameter through which the programmer can select either a line plot or a filled area plot.

To make the WRITE PLOT instruction easier to use at a higher level, a VAX command file called WPLOT was written. This file uses pre-defined values for several of the parameters associated with the WRITE PLOT instruction. A 512 point horizontal baseline was chosen, to be located across the center of the RAMTEK screen. This allows an equal number of pixels above and below the baseline to accommodate plot segments. The filled area baseline option was chosen, and plot segments are to be drawn from left to right. The programmer is responsible for providing some form of relative scaling. For this application the baseline was defined as relative zero. Plot segments with positive values should then fall above the baseline, and those with negative values should fall below it. The WPLOT command file operates on a previously created FORTRAN-formatted data file containing any 512 point function that the user wishes to plot on the RAMTEK display. Scaling occurs automatically. A background grid is drawn along with the plot. This grid defines a total plot area of 500 vertical pixels by 512 horizontal pixels. Each grid block is 50 pixels high by 64 pixels long. There are 5 blocks above and below the baseline in the vertical dimension, and 8 blocks along the baseline in the horizontal dimension. An alpha-numeric readout also appears at the bottom of the screen which indicates the maximum absolute value of the plotted function in either the positive or negative direction, whichever is greater. Scaling factors are computed such that the maximum absolute value in either the positive or negative direction is always a full scale deflection. Each grid block in the vertical direction, above or below the baseline would then represent one fifth of the maximum absolute value. Thus it is easy to evaluate or assess relative maxima or minima at a glance for any plot. To use the WPLOT command file the operator types @ WPLOT FILENAME, where "FILENAME" is the name of a data file containing the 512 point function to be plotted. No prompts appear at the operator's terminal, and the desired plot, along with the grid overlay and alpha-numeric annotation appear immediately on the RAMTEK screen. The only restriction on the FORTRAN data-file containing the function is that each point must be a 16-bit integer word.

The WPLOT command file executes a VAX FORTRAN program which computes scaling factors, generates end-points for the grid overlay, and generates ASCII character data for the alpha-numeric annotation. This data is then sent to CAP memory VIA the PSPIO utility. A CAP assembly language program is then loaded which utilizes the WRITE PLOT, WRITE VECTOR, and WRITE TEXT, instructions, along with the various coded parameters, to send all the required data to the RAMTEK screen. The CAP program is executed through a command from a separate FORTRAN routine incorporating the PSPIO utility.

(d) WIMAGEU and WIMAGEL

The RAMTEK WRITE IMAGE instruction accepts user-defined coordinates which specify a rectangular area of the screen, followed by a data stream containing the actual image pixel elements to be written. The position of a particular pixel in the data stream determines the screen address, and the intensity of that particular pixel is determined by a pre-loaded Video Look-Up Table (VLT). An 8-bit datum could represent up to 256 different intensity levels or "shades of grey". A single WRITE IMAGE instruction can transmit up to 32768 words to refresh memory. However, since the refresh memory for a 512 X 512 pixel display contains 262, 144 words, many WRITE IMAGE instructions are required to completely fill the CRT screen with image data. Use of the WRITE IMAGE instruction requires that each data element to be sent to the refresh memory be stored in a 16-bit CAP memory word. However, only the lower 8 bits of each word will actually be transmitted to refresh memory.

To provide a practical high level implementation for the RAMTEK WRITE IMAGE instruction, it is assumed that intensity data for a 512 X 512 pixel image is already resident in the MRSP Bulk Memory; and that the user is looking for an easy way to display it on the RAMTEK screen. As previously indicated, a 512 X 512 pixel image would require a block of Bulk Memory storage containing 262, 144 words. However, since the Bulk Memory is composed of 32-bit words, it would actually be possible to store two 512 X 512 pixel images in the same memory block. One image could reside in the upper 16 bits, and the other in the lower

16 bits. Based on these assumptions, two command files were written for the VAX called WIMAGEU and WIMAGEL. Each of these files writes a complete 512 X 512 pixel image from MRSP Bulk Memory directly to the RAMTEK display screen. WIMAGEU assumes that the image is resident in the upper 16 bits of Bulk Memory, while WIMAGEL assumes that it is resident in the lower 16 bits. To use these files from a standard computer terminal the operator simply enters @ WIMAGEU or @ WIMAGEL, depending upon what image he wishes to display. No prompts appear on the user's terminal, and the desired image is immediately transmitted to the RAMTEK screen.

The WIMAGEU and WIMAGEL command files load and execute a CAP assembly language program and an IOC assembly language program. The CAP program inputs image data from the Bulk Memory in blocks of 512 pixels, and scales it to insure that no pixel has a value greater than 255. This is the maximum intensity level that can be displayed on the RAMTEK CRT. Since the required scaling factor depends on the maximum intensity value present in the image, the entire 512 X 512 image must be searched prior to display. The CAP computes the scaling factor and inputs the image a second time in blocks of 512 pixels. It then implements the WRITE IMAGE instruction to send each block to the RAMTEK. A 512 pixel block represents one "line" of data on the RAMTEK screen. This process is repeated 512 times to generate the complete image. The IOC program moves image data from Bulk Memory to CAP memory in blocks of 512 pixels. Both the WIMAGEU and WIMAGEL command files load the same CAP program. However, different IOC programs are loaded, depending upon whether the image is to come from the upper or lower 16 bits of Bulk Memory. The CAP and IOC assembly language programs are initially executed by commands from a FORTRAN routine incorporating the PSPIO utility.

(e) WLOOKUP

All image data transmitted to the refresh memory VIA the RAMTEK WRITE IMAGE instruction is mapped through a Video Look-Up Table (VLT) prior to actual CRT display. The VLT defines the functional correspondence between data values stored in refresh memory and the actual grey scale intensity that is generated for each pixel. Use of a

look-up table enables the user to modify a stored image in a predictable manner, such as enhancing certain intensity levels, suppressing others, etc. The VLT is physically located in the RAMTEK Controller and consists of 2048 words, each 13 bits in length. As a physical device, the VLT has been flexibly designed to serve a variety of possible refresh memory/display configurations. The specific configuration used for the MRSP is based on a refresh memory size of 512 X 512 X 8 bits, which limits stored values to the integer range of 0-255. With this limitation, only the first 256 locations in the VLT have any practical significance, and values stored at higher locations are effectively ignored. The RAMTEK LOAD AUXILIARY MEMORY instruction is used to transmit values to the VLT. The Controller treats the VLT as an extension of refresh memory, and has assigned it an address of 8000 (hexadecimal). To use the LOAD AUXILIARY MEMORY instruction, the programmer follows the coded instruction with the hexadecimal address of the VLT, and then a 16-bit word containing the byte count, i.e., twice the actual number of values to be loaded into the VLT. This sequence is then followed by the actual values themselves. Intensity values generated by any desired function (linear, exponential, logarithmic, etc.) may be loaded into the VLT. In a practical sense, the values stored in refresh memory can actually be viewed as look-up table addresses. For each pixel to be displayed the RAMTEK Controller will access a specific location in refresh memory, which necessarily contains some integer number in the range 0-255. This number is then used as a vector to the VLT. Whatever value is stored at that location in the VLT becomes the actual grey scale intensity mapped on the CRT. If a straight linear function is loaded into the VLT (i.e., zero is stored at location zero, one is stored at location one, etc.) then the intensity values stored in refresh memory will be exactly reproduced on the CRT screen. If any other function is loaded into the VLT, some or all of the values in refresh memory will be modified prior to display. If no function is loaded (i.e., all zeros in the VLT) then the entire screen will remain black (dark) regardless of the values in refresh memory.

To provide a practical high level implementation of the RAMTEK video look-up table, a VAX command file called WLOOKUP was written. This command file operates on a previously created FORTRAN-formatted data

file containing any 256 point function that the user wishes to load. To execute the command file from a standard computer terminal the user simply types @ WLOOKUP FILENAME, where "FILENAME" is the name of a 256 point data file containing the desired function. This causes the video look-up table in the RAMTEK Controller to be directly loaded from the data file. No prompts appear at the user's terminal, and the data file is preserved.

The WLOOKUP command file executes a VAX FORTRAN program which reads the specified data file and sends the values directly to CAP memory VIA the PSPIO utility. A CAP assembly language program is then loaded which utilizes the RAMTEK LOAD AUXILIARY MEMORY instruction to send the data to the look-up table. The CAP program is initially executed by a command from a FORTRAN program incorporating the PSPIO utility.

(f) WCURSOR

The RAMTEK cursor appears on the display screen as a cross with the center element missing. It is located within a 14 X 14 pixel block, and may be positioned anywhere on the screen by means of a manual joystick control. Pertinent RAMTEK instructions relating to cursor operation are WRITE CURSOR STATE, READ CURSOR STATUS, and SENSE PERIPHERAL STATUS. These instructions allow the programmer to respectively generate the cursor at any desired point on the screen, read cursor position, and sense cursor status. Use of these instructions at the bit level is quite complex since the programmer must set a prefetch bit prior to reading cursor status or position, and then clear it before sending the next instruction. The prefetch bit controls the direction of transfer on the RAMTEK interface. Once it has been generated (initialized), the cursor can be moved to any location by means of the joystick. The joystick provides two modes of operation - TRACK and ENTER. In the TRACK mode, host processor interrupts are continuously generated as the cursor is being moved. This enables a program running in the host processor to dynamically monitor or "track" cursor status and position. In the ENTER mode, interrupts are only generated when the joystick operator depresses a momentary action switch. Each interrupt generated in this manner enables the host processor to monitor cursor

status and position. In the MRSP configuration, the RAMTEK host processor is the CAP; and an assembly language program running in the CAP continuously monitors cursor status. When an interrupt is generated in either the joystick TRACK or ENTER mode, the current cursor coordinates are read, converted to ASCII characters, and sent to the supplemental ONTEL display. An alpha-numeric message is also generated which indicates whether the coordinates were generated in the TRACK or ENTER mode.

To provide a practical high level implementation of the RAMTEK cursor, a VAX command file called WCURSOR was written. This file enables the user to initialize the cursor directly from the keyboard of a standard computer terminal. To execute this file, the user simply types @ WCURSOR. A message then appears on the user's terminal reminding him to power up the supplemental ONTEL display and put it in the EXECUTE mode. When this has been accomplished the user types a carriage return. The cursor then appears near the center of the RAMTEK screen, and is ready for manipulation by the joystick. In the joystick TRACK Mode, X and Y position coordinates will be continuously displayed on the ONTEL screen as the cursor is moved. In the ENTER mode, X and Y position coordinates will be sent to the ONTEL each time the user depresses the momentary action switch.

The WCURSOR command file loads the CAP assembly language program that monitors cursor status and position. This program utilizes the RAMTEK WRITE CURSOR STATE instruction to initially generate the cursor, and then implements the READ CURSOR STATUS AND SENSE PERIPHERAL STATUS instructions in a continuous loop. The CAP program is initially executed by a command from a FORTRAN routine utilizing the PSPIO utility.

(g) RAMCLEAR

The RAMCLEAR command file clears the RAMTEK CRT screen. To use this file the operator enters @ RAMCLEAR from the keyboard of a standard computer terminal. No prompts are generated at the user's terminal, and the RAMTEK CRT screen is immediately cleared. RAMCLEAR loads a CAP assembly language program which sends a MASTER

CLEAR to the RAMTEK Controller. The CAP program is executed by a command from a VAX FORTRAN routine incorporating the PSPIO utility.

(h) NEGATE

The NEGATE command file can be used to change the polarity of vector and text graphics currently on the RAMTEK screen. In normal operation white graphics are displayed on a dark background. The NEGATE command file will reverse the polarity of background and graphics so that dark graphics are displayed on a light background. To use this file the operator enters @ NEGATE from the keyboard of a standard computer terminal. No prompts are generated at the user's terminal, and polarity reversal occurs immediately with regard to the currently displayed graphics. The NEGATE command file loads a CAP assembly language program which sends a value of 255 (hexidecimal FF) to the first location of the RAMTEK video look-up table. This location controls the polarity of displayed graphics. The CAP program is executed by a command from a VAX FORTRAN routine incorporating the PSPIO utility.

c. Summary

The VAX command files described in the previous sections represent a high level implementation of the basic bit-level RAMTEK instruction set. They are easy to use, and can be executed from the keyboard of a standard computer terminal. Each file is designed to operate as an independent module, and the user can execute them as required in any sequence. If, for example, the user wishes to analyze a variety of look-up table enhancements on an image stored in bulk memory, he can successively invoke the WLOOKUP and WIMAGE command files. Data file inputs for WLOOKUP are easily created with VAX FORTRAN. Similarly, it is a straightforward matter to overlay graphics and/or a cursor on a displayed image using the WTEXT, WVECTOR and WCURSOR command files. The WPLOT command file is especially handy for plotting functions in many types of analysis work. Normally the original analysis routines are written in VAX FORTRAN, and only a few additional statements are required to save the results in data files. These functions can then be plotted on the RAMTEK display with a simple invocation of the WPLOT command file. Finally, the RAMCLEAR file is included as a matter of convenience. It

enables the display to be cleared by a simple keyboard command. However, the same function could be accomplished by walking over to the display and depressing the MASTER CLEAR switch.

2. DEMONSTRATION PROGRAM

To demonstrate the signal processing capabilities of the MRSP as a complete system, a two-dimensional Fast Fourier Transform (FFT) processing operation was developed. The FFT was selected because it can be directly related to many practical signal processing applications. For example, one approach to generating very high resolution spatial radar images utilizes FFT processing in two dimensions (range and azimuth) to perform pulse compression and doppler phase compression respectively. FFT's are also useful in spectrum analysis operations to specify the response characteristics of various filtering functions.

The demonstration program utilized all components of the MRSP system. Actual two-dimensional FFT computations were performed by the VAP. The CAP provided data scaling and control of the output display system, which was used to verify final results. The general purpose host computer was used to generate the initial input function data, and also the sine/cosine coefficient table required for FFT computations. The IOC provided control of the bulk memory, a multi-faceted task which included the acceptance of initial input data from the host, the transmission of intermediate data to the VAP, and the transmission of final output data to the CAP and RAMTEK.

Two versions of the two-dimensional FFT were written. The first version was a 16 X 16 point transform. This was later expanded to a 512 X 512 point transform when the final program was written. The program structure is essentially identical for both versions. Only some minor coding changes are required to modify instruction parameters which designate the number of points on which to operate. One advantage of initially working with a 16 X 16 point transform is that only 256 data memory locations are required. This enables the entire data memory to be easily printed on a single sheet of paper or displayed on the terminal screen for a quick inspection during program development. By contrast,

a 512 X 512 point transform requires 262, 144 memory locations, and only small portions of it can be displayed and inspected in a single glance.

The demonstration program generated a three-dimensional pulse of constant amplitude as the input function. The two-dimensional Fourier Transformation of this function should be a three-dimensional Sine X/X surface, with the main lobe positioned in the center of the surface. To verify the theoretical results, the output was presented in intensity modulated form on a 512 X 512 picture cell (pixel) cathode ray tube (CRT) monitor.

a. Function Description

A two-dimensional FFT is developed from a series of one-dimensional FFT's. A one dimensional FFT is a function of one variable, and it operates on an input function of N points, where N is the FFT length. N is usually specified as a power of 2. Figure 5(a) is a pulse of constant amplitude (A), which is used as an input function to a one-dimensional FFT. The pulse width is X points, and the total input function length is N points. When a discrete one-dimensional FFT and detect operation is performed on a pulse of constant amplitude, the result is a Sine X/X function (Reference 1). This is shown in Figure 5(b). The number of sidelobes is determined both by the number of samples in the pulse and the total number of samples in the FFT. Figure 6 shows the results for a pulse width of 16 samples and a total FFT length of 512 samples.

In a strict mathematical sense, a one-dimensional FFT is a function of only one variable. However, two real dimensions (amplitude and length) are required to describe the input function (pulse) on which the one-dimensional FFT is to operate. Likewise, two dimensions are required to describe the Sine X/X output. For computer implementation, it is convenient to represent the input function length dimension with a one-dimensional memory array of N locations. The other real dimension (amplitude) can then be determined by assigning values to various locations in the array. Any desired input function to a one-dimensional FFT can be described in this way. For the pulse of Figure 6, the first 16 array location have a value of A, and the remaining 496 locations have a value of zero.

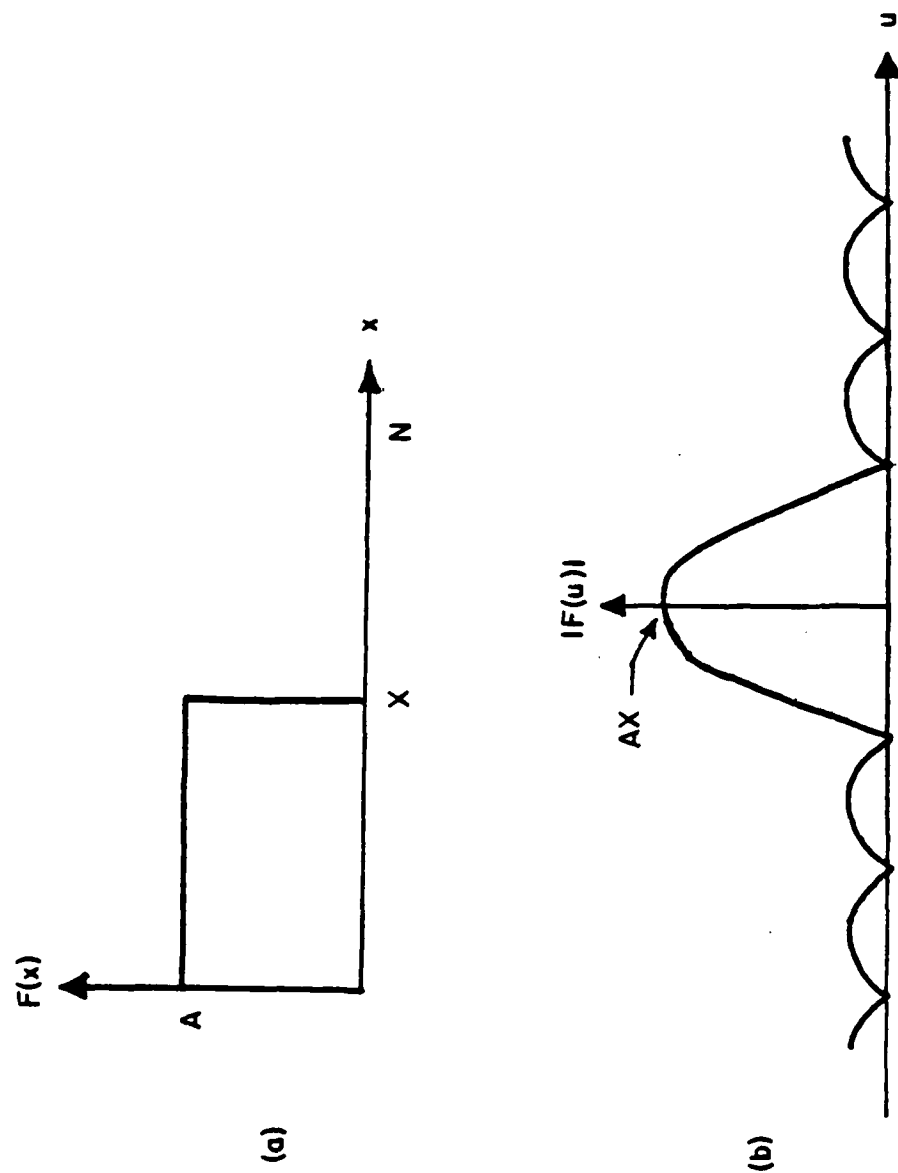


Figure 5. A Pulse Function and Fourier Transform

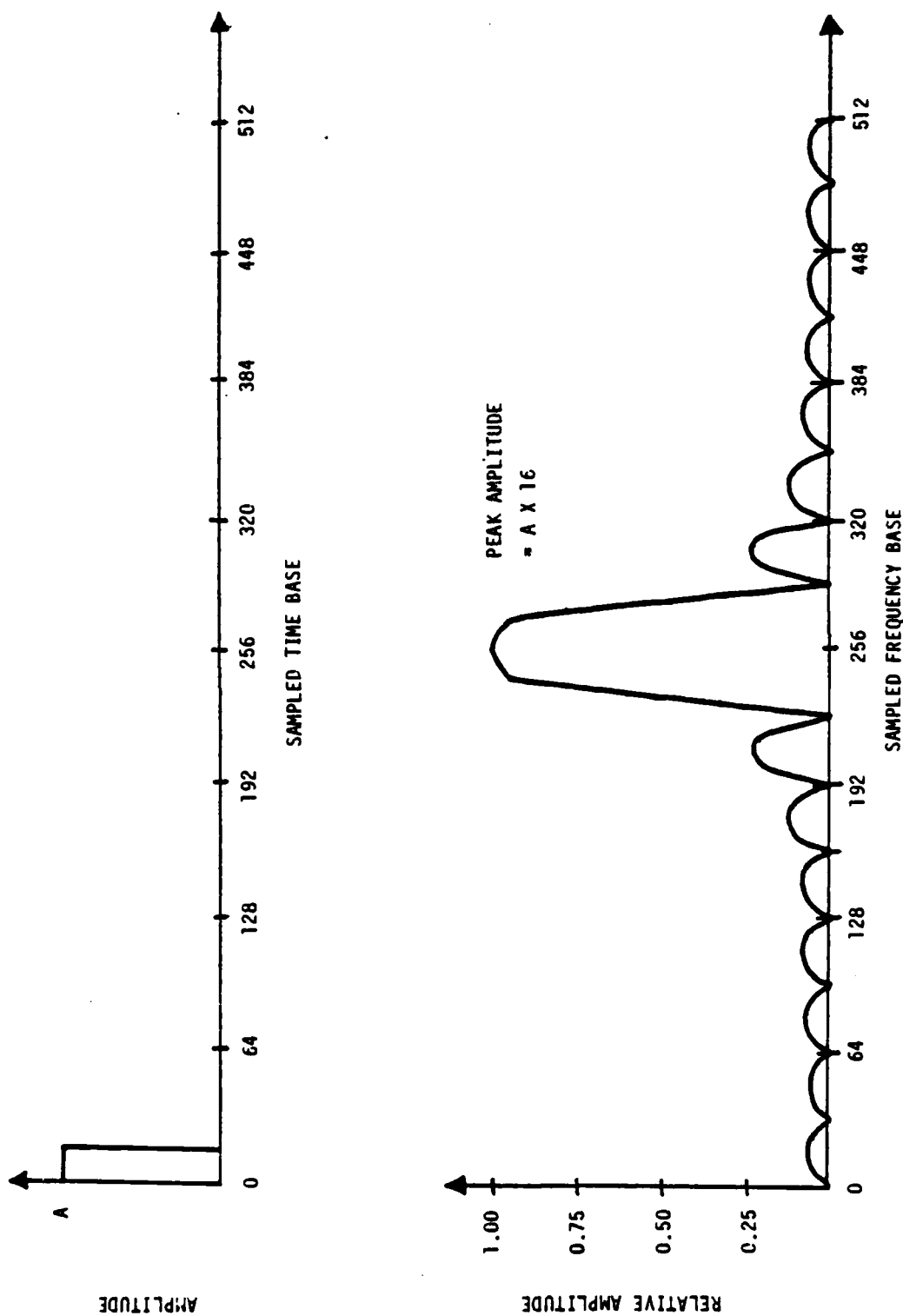


Figure 6. Sampled Fourier Transform Pair

A one-dimensional FFT can be expanded to a two-dimensional FFT by making it a function of two variables. In this case, the input function on which the FFT is to operate would be described by three real dimensions (length, width, and amplitude). The two-dimensional pulse of Figure 6 would then become a three-dimensional pulse. This is represented by the rectangular solid of Figure 7(a). A three dimensional function can be described by a two-dimensional computer memory array of $N \times N$ locations to represent length and width. The amplitude dimension is then represented by assigning values to the various array locations.

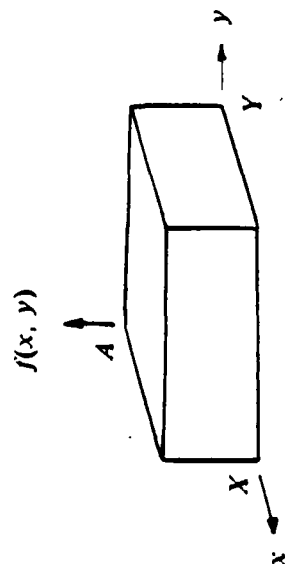
A two-dimensional FFT is implemented in two steps, each consisting of a series of one-dimensional FFT's. The first step is to compute N separate one-dimensional FFT's along one axis of the input function array (i.e., either the rows or the columns). The second step is to compute N more separate FFT's along the other dimension of the input array. All values in the input array are complex, i.e., they have a real component and an imaginary component. Each of the FFT's in the second step is followed by a detect operation ($\sqrt{\text{REAL}^2 + \text{IMAGINARY}^2}$) to arrive at the final result. For this demonstration program, the input function was a three-dimensional pulse of constant amplitude, and the desired output was a three-dimensional Sine X/X function similar to that of Figure 7(b).

b. IMPLEMENTATION

The basic VAP FFT instruction is a one-dimensional operation of N points. For the demonstration program, N is chosen to be 512 points. This is purely a matter of convenience, and is based on the fact that the available RAMTEK display system is configured for 512 X 512 pixels. The VAP can handle any FFT length up to 4096 points. The two-dimensional FFT will be developed in two steps as previously indicated. Each step consists of 512 one-dimensional FFT's of 512 points each.

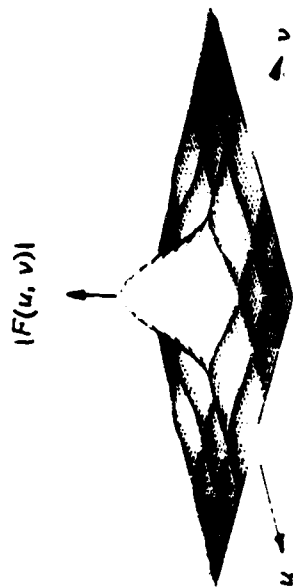
The input function is a three-dimensional pulse of constant amplitude. It is represented in computer memory by a 512 X 512 element complex array. The imaginary component of all elements in the array is set to zero. The real component for a 16 X 16 block of elements in the

THREE-DIMENSIONAL PULSE



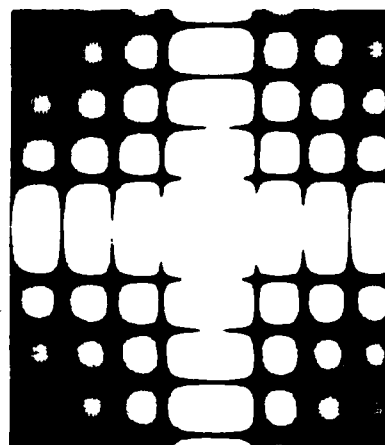
(a)

THREE-DIMENSIONAL FFT



(b)

INTENSITY REPRESENTATION FOR
THREE-DIMENSIONAL FFT



(c)

Figure 7. Three-Dimensional Representations

near corner of the array (i.e., rows 1-16 and columns 1-16) is set to a constant amplitude value of 1400. This function is generated in the VAX 11/750 host computer and transmitted to bulk memory via IOC Port #4. A 262, 144 element block of bulk memory is reserved for this purpose in the IOC program. The VAX then generates the 512 point sine/cosine coefficient table required for FFT computations. This is a complex data table in which the cosine coefficients constitute the real component and the sine coefficients constitute the imaginary component. The table is transmitted directly to VAP Data Memory #5 via the PSPIO support software utility routine. At this point the FFT operations commence. The IOC transmits a 512 element block of data to the VAP via IOC Port #5. The VAP computes a 512 point one-dimensional FFT and returns the results to the same locations in bulk memory via Port #0. The bulk memory address counter is appropriately updated and the next 512 element block of data is sent to the VAP, processed, and returned. This sequence continues, with the IOC "ping-ponging" between Port #5 and Port #0 until all 512 blocks of 512 elements each have been operated on by the VAP. The first step of the two-dimensional FFT operation has then been completed. All 512 rows of the input array have been processed. At this point a "corner turn" is implemented in the IOC. The second step of the operation requires all 512 columns to be processed in the same manner as the rows. The "corner turn" involves the generation of a different sequence of bulk memory addresses. Instead of accessing the data in sequential blocks of 512 as was done for the rows, the bulk memory address counter is now incremented by 512 after each memory access. When a complete column has been accessed, the memory address counter is decremented by 262, 144 and then incremented by one. This sequence continues until all 512 columns have been operated on by the VAP. The IOC "ping-pongs" between Port #5 and Port #0 as before. At this point a complete two-dimensional FFT has been computed. However, to derive the desired output (three-dimensional Sine X/X function), a detect operation is required after the second group of FFT's. This operation is performed in the VAP using the "Detect" instruction. The Detect instruction implements a very fast approximation of $\sqrt{\text{REAL}^2 + \text{IMAGINARY}^2}$. This operation could be performed in a separate process after the FFT's by moving the transformed data from bulk memory to VAP and then back to bulk memory again via Port #5 and Port #0 as

before. However, it is more efficient to combine the detect operation with the second set of FFT's. In this implementation, each of the 512 column FFT's is followed by a detect operation before the data is returned to bulk memory. This was the approach used in the demonstration program.

After the FFT/Detect operations have been completed, the IOC moves the data from bulk memory to CAP for presentation on the RAMTEK display system. This is accomplished sequentially in batches of 512 elements via IOC Port #3. The CAP scales each group of data as it is received and transmits it to the display CRT as a 512 element "line". When all 512 lines have been transmitted, the display screen is filled. The screen contains 262, 144 pixels, so there is a one-to-one correspondence with the transformed data in bulk memory. In addition to data scaling, the CAP has some other responsibilities. It generates and transmits the grey scale lookup table to the RAMTEK display system. It also processes the interrupts and controls the "handshaking" operations required to increment the RAMTEK line counter. The lookup table determines the intensity modulation on the RAMTEK display. For this demonstration program, a linear table is used which has values from 0 - 255. A pixel value of zero will be displayed as black, and a pixel value of 255 will be displayed as a bright white. Other pixel values will result in intermediate grey levels. The data scaling operation performed in the CAP insures that all pixels transmitted to the RAMTEK display will have values in the range 0 - 255. The final output display presentation is an intensity modulated Sine X/X function similar to Figure 7(c). Detailed listings of the VAP, CAP, IOC, and VAX programs for this demonstration are incorporated as Appendix A.

Only one VAP is required for the demonstration program which has been described, although the MRSP is actually configured with two VAP units. These are termed VAP1 and VAP2. The listings in the appendix are for VAP1. The same program was run in VAP2 with identical results. To use VAP2 a few coding changes are required in the IOC and VAX host programs. IOC Port #6 and Port #2 are used to communicate with VAP2, while Port #5 and Port #0 were used to communicate with VAP1. Also, different IOC flags are used to control the VAP2 Input/Output buffers.

The VAP1 and VAP2 programs themselves are, of course, identical. A one line change is required in the host VAX FORTRAN program to send the sine/cosine coefficient table to VAP2 rather than to VAP1. No changes are required in the CAP program to display VAP2 processed data.

Finally, it is worth noting that some cautions regarding amplitude scaling should be observed when programming FFT operations in the VAP. The VAP implements fixed-point integer arithmetic. When performing complex number operations, it treats the real and imaginary components as individual 16-bit two's complement quantities. Values for each component are thus limited to a dynamic range of -32,768 to +32,767. If saturation is allowed to occur in either the positive or negative direction, inaccurate or indeterminate results will occur. This is the programmer's responsibility to prevent, and no warning messages are generated. When contemplating FFT operations, it is recommended that the programmer perform a rough scaling analysis to estimate the maximum amplitudes to be expected. The FFT instruction is performed in stages, and amplitude values increase (accrue) with each successive stage. The number of stages performed is directly related to the total number of points in the FFT (previously denoted as N). The value of N is limited to a power of 2, and the number of stages is the actual exponent of 2. In the demonstration program, for example, each one-dimensional FFT is 512 points long. This is equivalent to 2^9 . Therefore, 9 stages are implemented for each one-dimensional FFT. The FFT instruction format allows each stage to be individually scaled. When a stage is scaled, all amplitudes are divided by a factor of two. Returning to the demonstration program, it will be recalled that the initial three-dimensional input pulse was set to a constant amplitude of 1400. The length and width dimensions were 16. The actual pulse thus consisted of 256 points of amplitude 1400. All other points in the input function were set to zero. From (Reference 1), it is noted that the maximum value to be expected from the Fourier Transformation of a constant amplitude pulse is equal to the amplitude of the pulse times the total number of points in the pulse. For the demonstration program this is $256 \times 1400 = 358,400$. Since this value is much larger than the saturation level, scaling is required. In this program, the last 4 stages in the second set of FFT's were scaled. The overall effect was to reduce the maximum expected

amplitude by a factor of 16. There are no fixed rules for determining what stages should be scaled. Intuitively it would appear better to scale the later stages in an FFT because larger values have accrued. Scaling smaller values in the early stages may cause unnecessary distortion to be carried through the entire process. Also, it appears that greater accuracy could be achieved by utilizing as large a portion of the dynamic range as possible without actually allowing saturation to occur.

c. TIMING CONSIDERATIONS

The VAP has sufficient hard-wired arithmetic to perform a full complex butterfly multiply and add in one clock cycle. A VAP clock cycle is 100 nanoseconds, or 0.1 microsecond. The butterfly operation is the basic computation required for FFT implementation in the VAP. As previously indicated, the FFT is performed in stages, where the number of stages is an integer exponent of 2. The number of clock cycles required for the VAP to compute a complete FFT is specified as $(N/2 + 3) \times (\text{NUMBER OF STAGES})$, where N is the total number of points in the FFT. A 512 point (9 stage) one-dimensional FFT would require 2331 clock cycles. This equates to about 233 microseconds. It is interesting to compare this computation time with that of a general purpose computer. A typical medium scale (32-bit) machine can implement a 32-bit integer multiply in 6.4 microseconds, and a 32-bit integer add in 0.4 microseconds. The time required for this machine to compute a complete 512 point FFT would depend on the specific algorithm implemented. One popular technique is the method of "successive doubling" (Reference 1). This technique divides an N-point transform into two parts, and computes two N/2 point transforms. The normal implementation limits N to a power of 2, and performs the computation in stages, where the number of stages is the exponent of 2. The number of multiplies required for this technique is $(N/2) \times (\text{NUMBER OF STAGES})$, and the number of adds is $N \times (\text{NUMBER OF STAGES})$. Using this particular algorithm and the above instruction times, a typical general purpose machine would require 16588 microseconds to compute a 512 point FFT. This is approximately 71 times slower than the VAP. The ratio remains about the same if the FFT size is increased to 4096 points (12 stages), which is the largest that can be

handled by the VAP. Assuming the ratio to be approximately valid, a complex signal processing problem requiring 10 seconds of VAP time would use 710 seconds or about 12 minutes on a typical general purpose machine.

When making timing comparisons, it is always difficult to calculate the execution time of a series of VAP instructions because the possibility of instruction overlap exists. This will decrease the actual execution time. Also, a general purpose machine normally operates in a time-shared environment, and there can be considerable variation in response time due to resource sharing and to system overhead. The estimates for a general purpose machine include only the theoretical FFT computation time, and do not include the overhead of moving data into and out of the various memory arrays. For comparison purposes, the VAP has an effective memory access time of 100 nanoseconds, while a typical general purpose machine has an effective memory access time of about 300 nanoseconds. Thus the VAP is about 3 times faster in this regard. The memory access factor becomes more significant with larger FFT's, and the overall effect is to significantly improve the VAP speed advantage.

3. PROGRAM FOR VERIFICATION OF DAG OPERATION

Verification of the Differential Address Generator (DAG) operation was obtained by programming the MRSP using information and an example provided by Westinghouse. The program implemented is a two-dimensional interpolation.

Verification required the use of three of the MRSP components: the VAP, the IOC, and the CAP. The VAP was used for the arithmetic operations necessary for the interpolation. The IOC was used to hold the input data, and to transfer the necessary data to the VAP for interpolation. Data for the DAG was provided by the CAP. Normally the CAP would dynamically create the necessary data for the DAG. In this case the CAP just transferred the previously calculated data to the DAG and provided operational control of the VAP and IOC. Also, the interpolation was accomplished in VAP1 and then accomplished in VAP2.

a. Functional Description

The DAG may be used to generate addressing for interpolations where the output (resampled) vector is linear or quadratic in relation to the input data. However, the DAG cannot be used where the slope of the output in relation to the input is not strictly increasing or strictly decreasing. Other applications of the DAG are possible, but they will not be discussed here.

Structure of the DAG consists of two relatively simple arithmetic units, AU1 and AU2. These are used to generate values which may be sent to one, two, or three double buffered tables. These tables are the Address Offset Table (AOT), Pointer Table 1 (PT1), and Pointer Table 2 (PT2). Values in the pointer tables are used to address VAP data memory locations; and values in the AOT are used to address the IOC bulk memory locations. Since the MRSP contains two VAPs, there are two double buffered pointer tables for each VAP. Words in the DAG register file (RF) determine which VAP will receive the pointer table values.

Inputs and configuration of the DAG are specified by the RF. The RF contains a control word, table looping values, AU1 input values, AU2 input values, and other control values. Determination of which values will be read by the DAG is provided by the control word. It also determines which tables will receive values.

Since the tables are double buffered, this allows very efficient use of the tables and fast access times. While one part of a table is being accessed, the other may be loaded by the DAG. This provides minimal restraint in table usage.

b. Program Operation

The program interpolates a 6 point vector from 19 input data vectors. Relation of the output to the input is linear, and the distance between output samples is constant. The operation involves an interpolation of points on the output vector from input data points in the vertical direction, and an interpolation of the desired output

locations from the previously obtained points (resampling along the output vector).

In order to do this, four vectors are taken from the input data. These vectors consist of points that are one element above and below the output, and points that are 2 elements above and below the output. A weighted average of these values is used to form points along the output vector. Next, points that are 1 and 2 elements to the right and to the left of the desired output points are extracted. A weighted average of these points determines the desired output point values.

The first part of the interpolation is performed by extracting four vectors from the IOC bulk memory and passing these to the VAP. The VAP will weight each vector and add them together to form points along the output vector. The second part of the interpolation is performed by extracting four vectors from the VAP data memory which consist of points from the previous interpolation results. These will be weighted and summed in the VAP to obtain the desired output vector.

The values of the weights are loaded into the VAP immediately after the VAP program is loaded. This includes the weights necessary for both interpolations.

c. DAG Requirement

Five DAG RF loads are required to implement the interpolation. The first load is used to create the values for PT1 and AOT. Loads number 2 through 4 are used to create AOT values. The fifth load creates values for both PT1 and PT2.

Values from the first PT1 load are used to access the weight values used in the first part of the interpolation. AOT values from the first load are used to transfer the vector consisting of all the points that are one element below the output vector to the VAP. Both of these tables are obtained from the first DAG RF load.

The second DAG RF load generates AOT values to transfer the vector of all points 2 elements below the output to the VAP. AOT values are also generated for the third and fourth RF loads, which transfer the vectors for 1 point above and 2 points above the output to the VAP. The values loaded into PT1 from the first RF load are accessed four times, once for each vector. However, four different weight sets in VAP data memory are accessed by the pointer table (PT1).

The fifth DAG RF load is used to generate both PT1 and PT2 values. These are used to extract and weight the vectors for the second part of the interpolation. PT2 is used in extracting the vectors and PT1 is used in accessing the weight values. Since both pointer tables may be accessed at the same time by the VAP, this allows very efficient operation. However, this creates some operational constraints, as the VAP must use PT1 in memory reference descriptor 1 only, and PT2 must be used in memory reference descriptor 3 only.

d. Considerations

The DAG provides an extremely useful tool in programming interpolations and related functions. Due to its required understanding level and lack of instructional material however, it is very difficult to program. This in part is due to the newness of the DAG component part of the MRSP. The logical conclusion is that the DAG is a very powerful tool, but its usefulness is currently limited by the support literature and lack of knowledge in the area of its uses.

4. SUPPORT SOFTWARE MODIFICATIONS

An on-going objective of the MRSP Integration Facility is to "streamline" the contractor-provided support software package whenever practical by incorporating some of the unique capabilities of the VAX 11/750 host computer. As previously indicated in Chapter 3, the support software package contains the modules required to assemble, link, load, debug, and execute useful programs for the MRSP. The support software is general in nature, and intended to be relatively independent of the user-provided host computer. However, some of these routines can be made to operate more efficiently by incorporating the unique features of a particular host.

Soon after delivery of the MRSP hardware, two particular support software operations emerged as obvious candidates for "streamlining". One of these was the tedious procedure required to assemble a CAP source program, and the other was the time-consuming BINGEN procedure, which converts an assembled CAP, VAP, or IOC program into a loadable binary module. Initial assembly of a CAP program requires three separate invocations of the File Manager support routine, several intermediate commands to supply data to and exit from the File Manager, an invocation of the linker/loader, and several additional commands to generate a hard copy of the source listing. This sequence of commands is difficult to remember, and frequently results in errors. By contrast, the procedure for generating a VAP or IOC assembly file and source listing is much simpler, essentially requiring only one command. Once a CAP, VAP, or IOC program has been assembled into the proper ASCII-octal format, the BINGEN routine is invoked to generate a binary module that can be loaded into the MRSP. The BINGEN routine supplied by the contractor was designed to operate on a 16-bit machine, such as the Digital Equipment Corporation PDP-11 Series. BINGEN will execute on the 32-bit VAX in "Compatibility Mode", however, it is relatively slow and inefficient. For example, BINGEN requires approximately 3 minutes to generate a binary load module from an assembled VAP, CAP, or IOC ASCII-octal format file.

In order to speed up the binary generation process, a native VAX routine called FASTBIN was developed. This routine completely bypasses BINGEN, and utilizes some of the unique capabilities of the VAX/VMS Operating System to derive binary data directly from an assembled ASCII-octal format file. As a result, FASTBIN generates exactly the same load module as BINGEN, but requires only about 3 seconds. A preliminary users manual has been developed for FASTBIN, which is included as APPENDIX B.

In order to "streamline" the CAP assembly process, an interactive command file called CAS was developed. CAS not only assembles a CAP source file, but at the programmers option, automatically invokes FASTBIN as well. In essence, CAS requires only one command to implement all the steps necessary to generate a loadable binary module from a CAP source program. In addition to FASTBIN, CAS incorporates the CAP loader, deassembler, and translator as options, and may thus be tailored to

perform a variety of operations. To use CAS, the programmer simply enters the name of the command file (CAS) followed by the optional parameter. A prompt then appears requesting the name of the CAP source file. Once this has been entered, the remainder of the procedure is totally automatic. The CAS command file has been incorporated as APPENDIX C.

5. CONCLUSIONS AND FUTURE ACTIVITIES

This interim report documents activities of the MRSP Integration Facility from project initiation February 1981 through December 1982. The actual MRSP hardware was delivered in March 1982. The primary accomplishments during this period include the development of a software package to integrate the output display system, some preliminary test programs to demonstrate the processing power of the MRSP, and several significant in-house improvements to the MRSP support software. Additionally, AFWAL/AARM personnel have gained a high degree of experience and programming proficiency with this relatively complex system.

The MRSP is basically a prototype machine. Experiments conducted during the course of this effort have verified that it has the capability to perform extremely high speed arithmetic operations on large blocks of data. In fact, the VAP operates most efficiently when the largest possible array lengths are used. Additionally, the MRSP has proven to be quite reliable from a maintenance standpoint.

The MRSP was purchased primarily to perform signal processing experiments relating to the generation and analysis of radar image data. However, very little work with actual image data was accomplished during the reporting period, primarily because of disk storage limitations on the VAX 11/750 host computer. A typical radar image consisting of 1024 X 1024 complex picture cells (pixels) would require about 4.3 megabytes of storage. Throughout most of the reporting period, the VAX was supporting several programs in addition to the MRSP, and could not conveniently accommodate such large files. Recently however, two large capacity disk drives (300 megabytes each) have been added to the VAX.

These will permit storage of large image files. During the next reporting period, some representative signal processing operations, including frequency filtering and detection, will be performed with actual radar image data to demonstrate the utility of the MRSP. Also during the next reporting period, a significant portion of the effort will be directed toward the Differential Address Generator (DAG) capabilities of the MRSP. The DAG is the MRSP hardware option that has the capability to generate high speed non-consecutive address sequences for the Bulk Memory or VAP data memory. This device has the potential to be extremely useful for certain nonlinear radar oriented interpolation operations. However, throughout most of the period covered by this report the DAG was plagued with persistent subtle hardware anomalies that inhibited its performance. A recent maintenance visit by the contractor has apparently corrected most of these problems. One operation of particular interest with regard to the DAG is the resampling of certain radar data from a polar format to a rectangular format, from which it can be more conveniently processed. The polar/rectangular resampling operation typically places a very high computational burden on a signal processor, and is usually accomplished at less than "real-time" rates. However, by virtue of the DAG, the MRSP has the potential to perform this operation much faster. During the next reporting period experiments will be designed to demonstrate this capability.

AFWAL-TR-83-1045

APPENDIX A
SAMPLE PROGRAM LISTINGS

SAMPLE PROGRAM LISTINGS

1. FFTDEMO.COM

This procedure executes a series of VAX/VMS Operating System level commands for a two-dimensional FFT demonstration.

2. F2VAP.VAP

This is a VAP assembly language program that performs computations for a 512 X 512 two-dimensional FFT.

3. F2DIOC.IOC

This is an IOC assembly language program that controls the Bulk Memory transfer required for a 512 X 512 two-dimensional FFT computation by VAP1. It also accepts input data from the VAX and transmits processed results to the CAP.

4. F2DCAP. CAP

This is a CAP assembly language program that scales processed data and transmits it to the RAMTEK Display.

5. F2DVAX.FOR

This is a VAX FORTRAN program that generates a 512 X 512 point input function for FFT operations. It also generates a 512 point sine/cosine coefficient table for the basic FFT computation.

6. IOCEXE.FOR

This VAX FORTRAN program uses the PSPIO utility to execute an operational program that has been loaded into the IOC.

7. CAPEXE.FOR

This VAX FORTRAN program uses the PSPIO utility to execute an operational program that has been loaded into the CAP.

1. FFT DEMO.COM

```

100 $ !
200 $ !
300 $ ! THIS PROCEDURE EXECUTES A SEQUENCE OF COMMANDS FOR A
400 $ ! TWO-DIMENSIONAL FFT DEMONSTRATION.
500 $ !
600 $ ! LOAD VAP,CAP,AND IOC BINARY MODULES.
700 $ !
800 $ PSPLDR WPA0:=F2DVAP
900 $ PSPLDR WPA0:=F2DIOC
1000 $ PSPLDR WPA0:=F2DCAP
1100 $ !
1200 $ ! RUN A VAX FORTRAN PROGRAM TO EXECUTE IOC MODULE.
1300 $ !
1400 $ RUN IOCEXE
1500 $ !
1600 $ ! EXECUTE VAX FORTRAN PROGRAM TO GENERATE INITIAL INPUT
1700 $ ! FUNCTION AND SINE/COSINE COEFFICIENT TABLE.
1800 $ !
1900 $ RUN F2DVAX
2000 $ !
2100 $ ! RUN A VAX FORTRAN PROGRAM TO EXECUTE CAP MODULE.
2200 $ !
2300 $ RUN CAPEXE
2400 $ !
2500 $ ! NOTE: VAP MODULE GENERATES ITS OWN EXECUTE COMMAND.
2600 $ !
2700 $ STOP

```

2. F2DVAP.VAP

```

100      TITLE F2DVAP
200
300      . . THIS PROGRAM PERFORMS A TWO-DIMENSIONAL FFT COMPUTATION.
400      . TWO SEPARATE PASSES THRU THE VAP ARE REQUIRED. EACH PASS
500      . PROCESSES 512 BLOCKS OF DATA IN WHICH EACH BLOCK CONTAINS
600      . 512 PIXELS.
700
800      . . DEFINE MEMORIES AND PARAMETERS
900
1000     INEND     EQU     1      . BIT SET TO INDICATE END OF TRANSFER
1100     BRN9      EQU     9
1200     VO1       EQU     8      . VIDEO OUTPUT 1 EQUIV TO MEM 8
1300     VO2       EQU     9      . VIDEO OUTPUT 2 EQUIV TO MEM 9
1400     VI1       EQU     0      . VIDEO INPUT 1 EQUIV TO MEM 0
1500     VI2       EQU     7      . VIDEO INPUT 1 EQUIV TO MEM 7
1600     NOEND     EQU     0      . BIT NOT SET - TRANSFER NOT COMPLETE
1700     INC       EQU     1
1800     K         DS      M5,0512
1900     K1        DS      M5,0513
2000     M1DATA    DS      M1,00  . VAP MEM 1 STARTS AT LOCATION 0
2100     M2DATA    DS      M2,00  . VAP MEM 2 STARTS AT LOCATION 0
2200     M3DATA    DS      M3,00  . VAP MEM 3 " " " "
2300     M4DATA    DS      M4,00  . VAP MEM 4 " " " "
2400     M5COEF    DS      M5,00  . VAP MEM 5 " " " "
2500     M6DATA    DS      M6,00
2600     M1UP      DS      M1,0512
2700     M2UP      DS      M2,0512
2800     M3UP      DS      M3,0512
2900     M4UP      DS      M4,0512
3000     M5UP      DS      M5,0512
3100     M6UP      DS      M6,0512
3200
3300     . . CLEAR DATA MEMS & MOVE IN DETECT CONSTANT
3400
3500     CLRM      N1024;M1;M2;M3;M4;M5;M6
3600     START     MOVECM  N1;K
3700
3800     . . MOVE IN 512 POINT SINE/COSINE COEFFICIENT TABLE FROM HOST
3900
4000     MOVECM    N512;M5COEF,,BRN9
4100
4200     . . PERFORM FIRST PASS FFT COMPUTATIONS
4300
4400     LOOP      MOVERO N512;VI1;M1DATA
4500             MOVERO N256;M1DATA+256;M2DATA
4600             BP $+1;1
4700             FFT N256;M5COEF;M1DATA;M2DATA;M3DATA;M4DATA;9;9;9
4800             MOVERO N256;M3DATA,,8;M1DATA
4900             MOVERO N256;M4DATA,,8;M1DATA+256
5000             MOVERO N512;M1DATA;VO1
5100             NEP LOOP;V512
5200
5300     . . PERFORM SECOND PASS FFT COMPUTATIONS
5400
5500     LOUP2     MOVERO N512;VI1;M1DATA
5600             MOVERO N256;M1DATA+256;M2DATA
5700             BP $+1;1
5800             FFT N256;M5COEF;M1DATA;M2DATA;M3DATA;M4DATA;9;9;9;S07
5900
6000     . . PERFORM DETECT OPERATION ON PROCESSED DATA
6100

```

2. F2DVAP.VAP (Continued)

6200	DET N256;K,I0;M3DATA,,8;M1DATA			
6300	DET N256;K,I0;M4DATA,,8;M2DATA			
6400	MOVERO N256;M1DATA;M2DATA+256			
6500	MOVERO N512;M2DATA;V01			
6600	REP LOOP2;V512			
6700	.			
6800	HALT			
6900	END			
7000	.			
7100	DATMEM	EQU	016	. DATMEM SELECT CODE IS 16(OCTAL)
7200	STADDR	EQU	0	. STARTING ADDRESS FOR MACRO PROGRAM
7300				. & DATMEM IS AT LOCATION 0
7400	MACRO	EQU	00	. MACRO PROGRAM SELECT CODE IS 00
7500	XQT	EQU	1	. EXECUTE BIT IS SET WITH A 1
7600	D32	EQU	1	. 32 BIT WORD TRANSFER IS SET WITH 1
7700	.			
7800	. EXECUTE VAP PROGRAM			
7900	.			
8000	FWABC STADDR;;MACRO,XQT			
8100	FWABC STADDR,1;;DATMEM,,D32			
8200	.			
8300	. DETECT CONSTANTS			
8400	0032405,0077777,0100000,0100000			
8500	END			

3. F2DIOC.IOC

```

100      TITLE F2DIOC
200      .
300      . . THIS PROGRAM SUPPORTS A 512 X 512 TWO-DIMENSIONAL FFT
400      . COMPUTATION WHICH IS PERFORMED BY VAP #1. PORT 4 ACCEPTS
500      . INITIAL DATA FROM HOST IN BLOCKS OF 512 PIXELS. PORTS
600      . 5 & 0 OPERATE IN PING-PONG FASHION TO MOVE DATA INTO AND
700      . OUT OF VAP IN BLOCKS OF 512 PIXELS. TWO PASSES THRU VAP
800      . ARE NECESSARY FOR A TWO-DIMENSIONAL FFT. A CORNER-TURN
900      . IS REQUIRED IN BULK MEM FOR SECOND PASS. PORT 3 SENDS THE
1000     . PROCESSED DATA TO CAP IN BLOCKS OF 512 PIXELS.
1100     .
1200     SCCTR 1,0
1300     BUFF  RES 262144
1400     EOT   EQU 1
1500     OUT   EQU 1
1600     IN    EQU 0
1700     LU    EQU 1018
1800     LL    EQU 1008
1900     UU    EQU 1118
2000     UL    EQU 1108
2100     DUMFLG EQU 9
2200     P4DONE EQU 6
2300     P5DONE EQU 8
2400     PCDONE EQU 7
2500     PMHOLD EQU 5
2600     .
2700     . . . PRIQ 4,5,0,3,1,2,6,7
2800     .
2900     . . . UNUSED PORTS
3000     . . . PORT 1
3100     . . . PORT 2
3200     . . . PORT 6
3300     . . . PORT 7
3400     HOLD  JUMP HOLD
3500     .
3600     . . . PORT 4
3700     . . . SETBA BUFF-1,IN
3800     LOUP4  INPOFS 1,1,LL
3900     . . . INPOFS 0,1,LU
4000     . . . REPEAT LOOP4,512,1
4100     . . . REPEAT LOOP4,512,2
4200     . . . SETF P4DONE
4300     . . . WAIT DUMFLG,OUT
4400     .
4500     . . . PORT 5
4600     . . . WAIT P4DONE,OUT
4700     . . . SETBA BUFF-1
4800     .
4900     . . . NOTE: FLAG 14 CONTROLS VAP #1 INPUT BUFFER
5000     .
5100     LOOPS  SETF 14
5200     . . . DELAY 8
5300     . . . OUTOFS 1,512,0,EDT
5400     . . . CLRF 14
5500     . . . DELAY 8
5600     . . . SETF P5DONE
5700     . . . WAIT P5DONE
5800     . . . REPEAT LOOPS,512,1
5900     .
6000     . . . CORNER-TURN FOR SECOND PASS THRU VAP
6100     . . . SETBA BUFF-512

```

3. F2DI0C.I0C (Continued)

```

6200  LOOP5A  SETF 14
6300          DELAY 8
6400          OUTOFS 512,512,0,,EOT
6500          CLRF 14
6600          DELAY 8
6700          INCBA 1
6800          DECBA 262144
6900          SETF P5DONE
7000          WAIT P0DONE
7100          REPEAT LOOP5A,512,1
7200          WAIT DUMFLG
7300  .
7400  .
7500          PORT 0
7600          SETBA BUFF-1,IN
7700  LOOP0  WAIT P5DONE
7800  .
7900  . . NOTE: FLAG 12 CONTROLS VAP #1 OUTPUT BUFFER
8000  .
8100          SETF 12
8200          DELAY 8
8300          INPOFS 1,512,0,,EOT
8400          CLRF 12
8500          DELAY 8
8600          SETF P0DONE
8700          REPEAT LOUPO,512,1
8800  .
8900  . . CORNER-TURN RETURNS DATA TO SAME LOCATIONS OF BULK MEM
9000          SETBA BUFF-512
9100  LOOP0A  WAIT P5DONE
9200          SETF 12
9300          DELAY 8
9400          INPOFS 512,512,0,,EOT
9500          CLRF 12
9600          DELAY 8
9700          INCBA 1
9800          DECBA 262144
9900          SETF P0DONE
10000         REPEAT LOOP0A,512,1
10100         SETF P0HOLD
10200         WAIT DUMFLG
10300  .
10400  .
10500         PORT 3
10600         WAIT P0HOLD,OUT
10700         SETBA BUFF-1
10800  LOOP3  CLRF 18
10900  LOOP3A  DELAY 8
11000  .
11100  . . NOTE: ONLY 64 PIXELS/TRANSFER FOR DMA CHAN #1
11200  . . FLAG 1H CONTROLS THIS TRANSFER
11300  .
11400          OUTOFS 1,64,LL,,EOT
11500          REPEAT LOUP3A,7,2
11600          DELAY 8
11700          OUTOFS 1,64,LL,18,EOT
11800          DELAY 8
11900          REPEAT LOUP3,512,1
12000  HOLD3  JUMP HOLD3
12100          END

```

4. F2DCAP.CAP

```

100  ;*A ASSEMBLE F2DCAP,F2DCAP 50
200  .
300  . . THIS PROGRAM ACCEPTS UNSCALED IMAGE DATA IN BLOCKS OF
400  . 512 PIXELS FROM BULK MEM THRU DMA CHAN #1. IT SCALES
500  . THE DATA AND SENDS IT TO RAMTEK THRU DMA CHAN #6. A
600  . TOTAL OF 512 BLOCKS ARE REQUIRED TO FILL THE RAMTEK
700  . DISPLAY. A LINEAR LOOKUP TABLE IS ALSO SENT TO RAMTEK.
800  .
900  . . DEFINE REGISTERS & PARAMETERS
1000 R1 EQU 1
1100 R2 EQU 2
1200 R3 EQU 3
1300 R4 EQU 4
1400 R5 EQU 5
1500 R6 EQU 6
1600 R7 EQU 7
1700 EQ EQU 2
1800 NE EQU 5
1900 .
2000 . . DEFINE INTERRUPTS 1 & 6
2100 MASK EQU \42
2200 .
2300 . . PROGRAM INITIALIZATION
2400 CCTR 1
2500 DSBL
2600 CLIR
2700 LIM R1,MASK
2800 SIC R1
2900 J BEGIN
3000 RES 24
3100 +INTAD1
3200 +INT1
3300 RES 8
3400 +INTAD6
3500 +INT6
3600 RES 82
3700 .
3800 . . GENERATE RAMTEK LOOKUP TABLE
3900 BEGIN LIM R1,DAT1
4000 ST R1,ADDR
4100 LIM R1,0
4200 BEG1 STI R1,ADDR
4300 IM ADDR
4400 AIM R1,1
4500 CIM R1,256
4600 JC NE,BEG1
4700 .
4800 . . CLEAR INTERRUPT FLAGS & RESET RAMTEK
4900 STZ DTRED6
5000 STZ DTRED1
5100 LIM R5,0
5200 LIM R1,0
5300 MO R1,\59
5400 LIM R1,\4200
5500 MO R1,\5A
5600 CLIR
5700 ENBL
5800 .
5900 . . LOAD LOOKUP TAB & WAIT FOR RAMTEK INTERRUPT
6000 SB 15,DTRED6
6100 LIM R1,LUTAB

```

4. F2DCAP.CAP (Continued)

6200	HO R1,\58
6300	LIM R1,\104
6400	HO R1,\59
6500	LIM R1,\2001
6600	HO R1,\5A
6700	HOLD1 L R1,DTRD6
6800	JC NE,HOLD1
6900	.
7000	.
7100	. . READ 512 WORDS FROM BULK MEM & WAIT FOR INTERRUPT
7200	. . STORE IN TEMP BUFFER
7300	LIM R4,0
7400	LIM R3,0
7500	INPUT SB 15,DTRD1
7600	LIM R1,BUFF1
7700	HO R1,\44
7800	LIM R1,\200
7900	HO R1,\45
8000	LIM R1,\11
8100	HO R1,\46
8200	HOLD2 L R1,DTRD1
8300	JC NE,HOLD2
8400	.
8500	. . SCALE IMAGE DATA BY FACTOR OF 128
8600	. . STORE IN OUTPUT BUFFER
8700	LIM R7,0
8800	LIM R1,BUFF1
8900	ST R1,ADDR1
9000	LIM R1,BUFF
9100	ST R1,ADDR
9200	LOOP LI R1,ADDR1
9300	SRA R1,7
9400	STI R1,ADDR
9500	IM ADDR1
9600	IM ADDR
9700	AIM R7,1
9800	CIM R7,512
9900	JC NE,LOOP
10000	.
10100	. . SEND 512 WORDS TO RAMTEK AND WAIT FOR INTERRUPT
10200	. . REPEAT ENTIRE SEQUENCE 512 TIMES
10300	SB 15,DTRD6
10400	LIM R1,RAM
10500	HO R1,\58
10600	LIM R1,\204
10700	HO R1,\59
10800	LIM R1,\2001
10900	HO R1,\5A
11000	HOLD3 L R1,DTRD6
11100	JC NE,HOLD3
11200	IM RAM+3
11300	AIM R3,1
11400	CIM R3,512
11500	JC NE,INPUT
11600	HALT
11700	J S
11800	.
11900	. . INTERRUPT SERVICE ROUTINES
12000	EVEN
12100	INT1 STZ DTRD1
12200	AIM R4,1

4. F2DCAP.CAP (Continued)

12300	ENBL	
12400	EXS INTAD1	
12500	.	
12600	EVEN	
12700	INT6 STZ DTRED6	
12800	AIM R5,1	
12900	ENBL	
13000	EXS INTAD6	
13100	.	
13200	. . RAMTEK INSTRUCTIONS TO LOAD LOOKUP TABLE	
13300	LUTAB DATA \300	
13400	DATA \8000	
13500	DATA \202	
13600	DAT1 RES 260	
13700	.	
13800	. . RAMTEK INSTRUCTIONS TO SEND IMAGE DATA TO DISPLAY	
13900	RAM DATA \A03	
14000	DATA \C0	
14100	DATA \0	
14200	DATA \0	
14300	DATA \1FF	
14400	DATA \1FF	
14500	DATA \0	
14600	DATA \400	
14700	BUFF RES 520	
14800	.	
14900	. . TEMP STORAGE FOR UNSCALED DATA	
15000	BUFF1 RES 520	
15100	.	
15200	. . INTERRUPT LINKAGE ADDRESSES	
15300	EVEN	
15400	INTAD1 RES 2	
15500	INTAD6 RES 2	
15600	.	
15700	. . INTERRUPT FLAGS	
15800	DTRED1 RES 1	
15900	DTRED6 RES 1	
16000	.	
16100	. . BUFFER POINTERS	
16200	ADDR RES 1	
16300	ADDR1 RES 1	
16400	END	

5. F2DVAX.FOR

```

0001      PROGRAM F2DVAX
          C
          C.....THIS PROGRAM SUPPORTS A TWO-DIMENSIONAL FFT COMPUTATION
          C      WHICH IS ULTIMATELY PERFORMED BY VAP #1. THE PROGRAM
          C      GENERATES THE INITIAL THREE-DIMENSIONAL PULSE FUNCTION
          C      AND SENDS IT TO BULK MEMORY. IT THEN GENERATES THE 512
          C      POINT SINE/COSINE COEFFICIENT TABLE AND SENDS IT TO
          C      VAP #1 DATA MEMORY
          C
          C
0002      IMPLICIT INTEGER*2 (B-Y)
0003      INCLUDE 'PSP$LIB:PSP$IOF.INC'
          * C
          * C --- DEFINE PSP I/O FUNCTION CODES.
0004      *      PARAMETER IOWALL = 1      !WRITE PASS ALL
0005      *      PARAMETER IORALL = 2      !READ PASS ALL
0006      *      PARAMETER IOWDR = 3      !WRITE DEVICE REGISTERS
0007      *      PARAMETER IORDR = 4      !READ DEVICE REGISTERS
0008      *      PARAMETER IOINIT = 5      !INITIALIZE DR11-BIF
0009      *      PARAMETER IOSYSC = 6      !SYSCRAH DMA CHANNEL BUS AND CAP
0010      *      PARAMETER IOXQT = 7      !EXECUTE CAP
0011      *      PARAMETER IOWRIT = 8      !WRITE DATA, GEN ALERT IF CBUS
0012      *      PARAMETER IOREAD = 9      !READ DATA, GEN ALERT IF CBUS
0013      *      PARAMETER IOWCON = 10     !WRITE, CONTINUATION OF PREVIOUS TRANSFER
0014      *      PARAMETER IORCON = 11     !READ, CONTINUATION OF PREVIOUS TRANSFER
0015      *      PARAMETER IOWHA = 12     !WRITE HARD ADDRESS, GENERATE ALERT WORDS
0016      *      PARAMETER IORHA = 13     !READ HARD ADDRESS, GENERATE ALERT WORDS
0017      *      PARAMETER MAXFUN = 13     !MAXIMUM LEGAL FUNCTION CODE
          * C
          * C --- DEFINE PSP I/O SUBFUNCTION MODIFIER BITS.
0018      *      PARAMETER IOTR32 = 256 !PERFORM 32-BIT TRANSFER
0019      *      PARAMETER IOTEST = 512 !ENABLE TEST MODE
0020      *      PARAMETER IOLOCK = 1024 !LOCKOUT ACCESS TO SLAVE BY OTHER CHANNELS
0021      *      INCLUDE 'PSP$LIB:UTYPES.INC'
0022      *      PARAMETER CAPTYP=1, IOCTYP=2, VAPTYP=3, BMTYP=4, CBPTYP=5
0023      *      INCLUDE 'PSP$LIB:UNITSX.INC'
0024      *      DATA UNTDSC
          *      A / 'VA','P1', ' ', 3, 2, 1, 3, 5*0,
          *      B / 'IO','C ', ' ', 2, 2, 2, 3, 5*0,
          *      C / 'CA','P ', ' ', 1, 0, 0, 0, 5*0,
          *      1 / 'VA','P2', ' ', 3, 2, 3, 3, 5*0,
          *      E / 'BU','LK', ' ', 4, 2, 4, 0, 5*0,
          *      F / 'MX','M ', ' ', 5, 4, 4, 4, 3, 1, 0, 1, 0,
          *      G / 'NU','LL', ' ', 9*0,
          *      H / 'NU','LL', ' ', 9*0/
          * C
          * C
0025      COMMON /UNTDSC/ DMODE
0026      INTEGER*2 UNTDSC(12,8)
0027      INTEGER*2 IA(512),IB(512)
0028      DOUBLE PRECISION AINC,AD,ANX,AAX
0029      COMMON /DRDEVX/ DUN,DRDEV(3,4)
0030      DIMENSION OUTPUT(1027)
0031      DATA TOPSP,FRMPSP,PSP,VAP,TMO,IOSTAT/1,2,1,1,0,0/
0032      DATA OUTPUT(1),OUTPUT(2),OUTPUT(3)/0,1024,"1016/
0033      DATA DRDEV
          *      A / 'WP','AU', ': ', 'WP','BU', ': ', 6* ' ' /

```

5. F2DVAX.FOR (CONTINUED)

```

C
C.....ASSIGN A VAX I/O CHANNEL
C
0034      CALL ASNOEV (DRDEV,5,DUNIT,IOSTAT)
0035      IF (IOSTAT.NE.0) GO TO 9999
C
C.....GENERATE THREE DIMENSIONAL PULSE FUNCTION
C
0036      NUMOUT=1024
0037      DO 100 I=4,1027
0038      100      OUTPUT(I)=0
0039      DO 101 I=5,35,2
0040      101      OUTPUT(I)=1400
0041      DO 105 I=1,512
0042      IA(I)=OUTPUT(2*I+3)
0043      IA(I)=IA(I)*(-1)**I
0044      OUTPUT(2*I+3)=IA(I)
0045      105      CONTINUE
C
C.....SEND PULSE FUNCTION TO BULK MEMORY
C
0046      DO 110 I=1,16
0047      110      CALL PSPIO(IOWRIT,DUNIT,UNTDSC(1,5),0,NUMOUT
1          ,OUTPUT(4),0,IOSTAT)
0048      DO 140 I=4,67
0049      140      OUTPUT(I)=0
0050      DO 150 I=17,512
0051      150      CALL PSPIO(IOWRIT,DUNIT,UNTDSC(1,5),0,NUMOUT
1          ,OUTPUT(4),0,IOSTAT)
C
C.....GENERATE FFT SINE/COSINE COEFFICIENTS
C
0052      AD=3.14159265
0053      AINC=AD/512
0054      DO 160 I=1,512
0055      AAX=DCOS(AD)
0056      ABX=DSIN(AD)
0057      AA=AAX*32768
0058      AB=ABX*32768
0059      AD=AD+AINC
0060      IA(I)=AA
0061      IB(I)=AB
0062      OUTPUT(2*I+2)=IB(I)
0063      OUTPUT(2*I+3)=IA(I)
0064      160      CONTINUE
C
C.....SEND FFT COEFFICIENTS TO VAP DATA MEMORY
C
0065      NUMOUT=1027
0066      CALL PSPIO(IOWRIT,DUNIT,UNTDSC(1,1),0,NUMOUT
1          ,OUTPUT(1),0,IOSTAT)
0067      9999      STOP
0068      END

```

6. IOCEXE.FOR

```

0001      PROGRAM IOCEXE
      C
      C.....THIS PROGRAM USES THE PSP10 UTILITY TO EXECUTE A BINARY
      C      MODULE THAT HAS BEEN LOADED INTO THE IOC.
      C
0002      IMPLICIT INTEGER*2 (B-Y)
0003      INCLUDE 'PSP$LIB:PSP10F.INC'
      * C
      * C --- DEFINE PSP I/O FUNCTION CODES.
0004      *      PARAMETER IOWALL = 1      !WRITE PASS ALL
0005      *      PARAMETER IORALL = 2      !READ PASS ALL
0006      *      PARAMETER IOWDR = 3      !WRITE DEVICE REGISTERS
0007      *      PARAMETER IORDR = 4      !READ DEVICE REGISTERS
0008      *      PARAMETER IOINIT = 5      !INITIALIZE DR11-BIF
0009      *      PARAMETER IOSYSC = 6      !SYSCRAH DMA CHANNEL BUS AND CAP
0010      *      PARAMETER IOXQT = 7      !EXECUTE CAP
0011      *      PARAMETER IOWRIT = 8      !WRITE DATA, GEN ALERT IF CBUS
0012      *      PARAMETER IOREAD = 9      !READ DATA, GEN ALERT IF CBUS
0013      *      PARAMETER IOWCON = 10     !WRITE, CONTINUATION OF PREVIOUS TRANSFER
0014      *      PARAMETER IORCON = 11     !READ, CONTINUATION OF PREVIOUS TRANSFER
0015      *      PARAMETER IOWHA = 12     !WRITE HARD ADDRESS, GENERATE ALERT WORDS
0016      *      PARAMETER IORHA = 13     !READ HARD ADDRESS, GENERATE ALERT WORDS
0017      *      PARAMETER MAXFUN = 13     !MAXIMUM LEGAL FUNCTION CODE
      * C
      * C --- DEFINE PSP I/O SUBFUNCTION MODIFIER BITS.
0018      *      PARAMETER IOTH32 = 256    !PERFORM 32-BIT TRANSFER
0019      *      PARAMETER IOTEST = 512    !ENABLE TEST MODE
0020      *      PARAMETER IOLOCK = 1024  !LOCKOUT ACCESS TO SLAVE BY OTHER CHANNELS
0021      INCLUDE 'PSP$LIB:UTYPES.INC'
0022      *      PARAMETER CAPTYP=1, IOCTYP=2, VAPTYP=3, BMTYP=4, CBPIYP=5
0023      INCLUDE 'PSP$LIB:UNITSX.INC'
0024      *      DATA UNTDSC
      *      A / 'VA', 'P1', ' ', 3, 2, 1, 3, 5*0,
      *      B / 'IO', 'C ', ' ', 2, 2, 2, 3, 5*0,
      *      C / 'CA', 'P ', ' ', 1, 0, 0, 0, 5*0,
      *      1 / 'VA', 'P2', ' ', 3, 2, 3, 3, 5*0,
      *      E / 'BU', 'LK', ' ', 4, 2, 4, 0, 5*0,
      *      F / 'MX', 'M ', ' ', 5, 4, 4, 3, 1, 0, 1, 0,
      *      G / 'NU', 'LL', ' ', 9*0,
      *      H / 'NU', 'LL', ' ', 9*0/
      * C
      * C
0025      COMMON /UNTDSC/ DMODE
0026      INTEGER*2 UNTDSC(12,8)
0027      INTEGER*2 FUNCWD(3)
0028      COMMON /ORDEVX/ DUN,ORDEV(3,4)
0029      DATA TOPSP,FRMPSP,PSP,VAP,IMO,IOSTAT/1,2,1,1,0,0/
0030      DATA FUNCWD/0,0,"100000"/
0031      DATA DRDEV
      A / 'WP', 'A0', ': ', 'WP', 'B0', ': ', 6* ' ' /
      C
      C.....ASSIGN A VAX I/O CHANNEL
      C
0032      CALL ASNDEV (ORDEV,5,DUNIT,IOSTAT)
0033      IF (IOSTAT.NE. 0) GO TO 9999
      C

```

AFWAL-TR-83-1045

6. IOCEXE.FOR (CONTINUED)

C.....EXECUTE IOC

0034	C	CALL PSPIU(IOWRIT,DUNIT,ONTDSC(1,2),0,3
		1 ,FUNCWD,0,IUSTAT)
0035	C	
0036	9999	STOP
		END

7. CAPEXE.FOR

```

0001      PROGRAM CAPEXE
      C
      C.....THIS PROGRAM USES THE PSPID UTILITY TO EXECUTE A BINARY
      C      MODULE THAT HAS BEEN LOADED INTO THE CAP.
      C
0002      IMPLICIT INTEGER*2 (B-Y)
0003      INCLUDE 'PSP$LIB:PSPIDF.INC'
      * C
      * C ---      DEFINE PSP I/O FUNCTION CODES.
0004      *      PARAMETER IOWALL = 1      !WRITE PASS ALL
0005      *      PARAMETER IORALL = 2      !READ PASS ALL
0006      *      PARAMETER IOWDR = 3      !WRITE DEVICE REGISTERS
0007      *      PARAMETER IORDR = 4      !READ DEVICE REGISTERS
0008      *      PARAMETER IOINIT = 5      !INITIALIZE DR11-BIF
0009      *      PARAMETER IOSYSC = 6      !SYSCRAH DMA CHANNEL BUS AND CAP
0010      *      PARAMETER IOXOT = 7      !EXECUTE CAP
0011      *      PARAMETER IOWRIT = 8      !WRITE DATA, GEN ALERT IF CBUS
0012      *      PARAMETER IOREAD = 9      !READ DATA, GEN ALERT IF CBUS
0013      *      PARAMETER IOWCON = 10     !WRITE, CONTINUATION OF PREVIOUS TRANSFER
0014      *      PARAMETER IORCON = 11     !READ, CONTINUATION OF PREVIOUS TRANSFER
0015      *      PARAMETER IOWHA = 12     !WRITE HARD ADDRESS, GENERATE ALERT WORDS
0016      *      PARAMETER IORHA = 13     !READ HARD ADDRESS, GENERATE ALERT WORDS
0017      *      PARAMETER MAXFUN = 13     !MAXIMUM LEGAL FUNCTION CODE
      * C
      * C ---      DEFINE PSP I/O SUBFUNCTION MODIFIER BITS.
0018      *      PARAMETER IOTR32 = 256    !PERFORM 32-BIT TRANSFER
0019      *      PARAMETER IOTEST = 512    !ENABLE TEST MODE
0020      *      PARAMETER IOLOCK = 1024   !LOCKOUT ACCESS TO SLAVE BY OTHER CHANNELS
0021      *      INCLUDE 'PSP$LIB:UTYPES.INC'
0022      *      PARAMETER CAPTYP=1, IOCTYP=2, VAPTYP=3, BMTYP=4, CBPTYP=5
0023      *      INCLUDE 'PSP$LIB:UNITSX.INC'
0024      *      DATA UNTDSC
      *      A / 'VA','P1', ' ', 3, 2, 1, 3, 5*0,
      *      B 'IO','C ', ' ', 2, 2, 2, 3, 5*0,
      *      C 'CA','P ', ' ', 1, 0, 0, 0, 5*0,
      *      1 'VA','P2', ' ', 3, 2, 3, 3, 5*0,
      *      E 'BU','LK', ' ', 4, 2, 4, 0, 5*0,
      *      F 'MX','M ', ' ', 5, 4, 4, 4, 3, 1, 0, 1, 0,
      *      G 'NU','LL', ' ', 9*0,
      *      H 'NU','LL', ' ', 9*0/
      * C
      * C
0025      COMMON /UNTDSC/ DMODE
0026      INTEGER*2 UNTDSC(12,8)
0027      INTEGER*2 FUNCWD(3)
0028      COMMON /DRDEVX/ DUN,DRDEV(3,4)
0029      DATA TOPSP,FRMPSP,PSP,VAP,IHO,IOSTAT/1,2,1,1,0,0/
0030      DATA FUNCWD/0,0,"100000/
0031      DATA DRDEV
      A / 'WP','AO', ': ', 'WP','BO', ': ', 6* ' ' /
      C
      C.....ASSIGN A VAX I/O CHANNEL
      C
0032      CALL ASNDEV (DRDEV,5,DUNIT,IOSTAT)
0033      IF (IOSTAT.NE.0) GO TO 9999
      C

```

7. CAPEXE.FOR (CONTINUED)

	C.....EXECUTE CAP	
	C	
0034		CALL PSPIO(IXQT,DUNIT,ONTOSC(1,3),0,3
		1 ,FUNCWD,0,IUSTAT)
	C	
0035	9999	STOP
0036		END

AFWAL-TR-83-1045

APPENDIX B

FASTBIN USERS MANUAL AND INSTALLATION GUIDE

0.0 PREFACE

0.1 MANUAL OBJECTIVES

THIS MANUAL DESCRIBES THE FASTBIN BINARY LOAD MODULE GENERATOR FOR THE WESTINGHOUSE PSPX+ SIGNAL PROCESSOR. THIS MANUAL IS DESIGNED PRIMARILY FOR REFERENCE ALTHOUGH IT IS SLIGHTLY TUTORIAL.

0.2 INTENDED AUDIENCE

THIS MANUAL IS INTENDED FOR READERS WHO ARE FAMILIAR WITH THE WESTINGHOUSE PSPX+ AND WHO UNDERSTAND THE PROCESS OF PROGRAM DEVELOPMENT FOR THAT SYSTEM. THE READER SHOULD ALSO BE FAMILIAR WITH THE VAX DCL COMMAND LANGUAGE AND WITH THE USE OF THE WESTINGHOUSE BINGEN BINARY LOAD MODULE GENERATOR.

0.3 STRUCTURE OF THIS DOCUMENT

THIS MANUAL CONTAINS 4 SECTIONS AND 2 APPENDICES.

1. SECTION 1 INTRODUCES WHAT FASTBIN IS USED FOR.
 2. SECTION 2 DESCRIBES THE THREE WAYS TO ACTIVATE THE FASTBIN.EXE IMAGE.
 3. SECTION 3 DESCRIBES THE FASTBIN OUTPUT.
 4. SECTION 4 CONTAINS INSTRUCTIONS ON CREATING CUSTOM FASTBIN HEADER FILES.
 5. APPENDIX A PROVIDES THE SYNTAX OF THE EXTENDED DCL FASTBIN COMMAND LINE.
 6. APPENDIX B DESCRIBES THE THREE WAYS TO INSTALL THE FASTBIN.EXE IMAGE INTO A VAX SYSTEM.
-

0.4 ASSOCIATED DOCUMENTS

FAMILIARITY WITH THE CONTENTS OF THE FOLLOWING DOCUMENTS WILL BE ESPECIALLY HELPFUL IN UNDERSTANDING THIS MANUAL:



1. THE VAX/VMS COMMAND LANGUAGE USER'S GUIDE. PAY PARTICULAR ATTENTION TO APPENDIX F: FOREIGN COMMAND FEATURE OF DCL.
2. THE VAX/VMS VERSION 3.0 UTILITIES REFERENCE MANUAL. PAY PARTICULAR ATTENTION TO THE CHAPTER ON THE DCL COMMAND DEFINITION UTILITY (SET COMMAND).
3. THE WESTINGHOUSE PSPX+ CAP PROGRAMMERS MANUAL (9RA8002H01).
4. THE WESTINGHOUSE PSPX+ IUC/VAP/CHANNEL BUS USER'S MANUAL (9RA8001H01).

1.0 INTRODUCTION

PROGRAM DEVELOPMENT FOR THE WESTINGHOUSE PSPX+ SIGNAL PROCESSOR STARTS WITH WRITING A SOURCE PROGRAM THAT IS DESIGNED TO RUN IN ONE OF EITHER THE CAP, VAP OR IUC PROCESSORS. THIS SOURCE PROGRAM IS THEN ASSEMBLED WITH THE APPROPRIATE ASSEMBLER TO PRODUCE AN ASCII OBJECT OUTPUT FILE (A .MAC FILE). NEXT, A BINARY LOAD MODULE IS CREATED USING EITHER THE BINGEN OR FASTBIN LOAD MODULE GENERATION UTILITIES AND THE OUTPUT (A .BIN FILE) IS LASTLY SPECIFIED AS INPUT TO THE PSPLDR UTILITY. PSPLDR LOADS THE EXECUTABLE FILE INTO THE APPROPRIATE PSPX+ COMPONENT PROCESSOR AND THE PROGRAM IS READY TO RUN. THIS MANUAL DESCRIBES HOW TO USE THE FASTBIN BINARY LOAD MODULE GENERATOR MENTIONED ABOVE IN ORDER TO PRODUCE LOAD MODULES ACCEPTABLE TO PSPLDR AND THE TARGET PSPX+ PROCESSORS.

2.0 FASTBIN COMMAND SYNTAX

IN ORDER TO PRODUCE ACCEPTABLE OUTPUT, FASTBIN ACQUIRES FROM THE USER THREE PIECES OF INFORMATION:

1. THE IDENTITY OF THE PSPX+ COMPONENT PROCESSOR FOR WHICH A LOAD MODULE IS BEING GENERATED (CAP, VAP OR IUC).
2. THE FILE NAME SPECIFICATION OF THE ASCII OBJECT CODE FILE (.MAC FILE) PRODUCED BY A PSP ASSEMBLER. HENCEFORTH, THIS SHALL BE CALLED THE SOURCE FILE.
3. THE FILE NAME SPECIFICATION FOR THE OUTPUT FILE (THE .BIN FILE TO BE PRODUCED).

THE MANNER IN WHICH FASTBIN ACQUIRES THIS INFORMATION IS DEPENDENT ON THE DCL COMMAND SYNTAX USED TO ACTIVATE THE

FASTBIN.EXE IMAGE. THE IMAGE ACTIVATION SYNTAX FORMS ARE DCL SRUN, DCL FOREIGN AND THE EXTENDED DCL SPASIBIN. THE FOLLOWING SECTIONS DESCRIBE EACH OF THESE IMAGE ACTIVATION METHODS. IT IS IMPORTANT TO NOTE HERE THAT ONLY ONE OF THESE IS USUALLY USED, WHICH ONE DEPENDS ON HOW THE LOCAL VAX SYSTEM MANAGER INSTALLED THE FASTBIN UTILITY.

2.1 DCL SRUN

FASTBIN MAY BE ACTIVATED USING THE DCL COMMAND SRUN IN THE FORMAT:

SRUN FASTBIN.EXE

IF ACTIVATED IN THIS MANNER, FASTBIN PROMPTS FOR:

1. 'HEADER FILE:' THE NAME OF A DATA FILE THAT DESCRIBES THE IDENTITY OF THE PSPAT COMPILER PROCESSOR FOR WHICH A LOAD MODULE IS BEING GENERATED. THE STANDARD RESPONSES ARE:

CAP.DAT - IDENTIFIES A CAP GENERATION.
IOC1.DAT - FOR IOC #1.
VAP1.DAT - FOR VAP #1.
VAP2.DAT - FOR VAP #2.
BM.DAT - FOR BULK MEM.
MXM.DAT - FOR ?

TABLE 1: STANDARD HEADER FILE NAMES.

IF A FILE TYPE IS NOT SPECIFIED IT WILL DEFAULT TO .DAT. THE DEFAULT DIRECTORY OF THE DATA FILES IS POINTED TO BY THE LOGICAL NAME 'FAST_DAT:'.

2. 'SOURCE FILE:' THE NAME OF THE ASCII OBJECT SOURCE FILE. THE DEFAULT FILE TYPE IS .MAC AND THE DEFAULT DIRECTORY IS THE CURRENT PROCESS DEFAULT DIRECTORY.

WHEN ACTIVATED VIA DCL SRUN, FASTBIN PROMPTS FOR AN OUTPUT FILE WITH THE SAME NAME AS THE SOURCE FILE, BUT A FILE TYPE OF .BIN.

EXAMPLE:

SRUN FASTBIN.EXE
HEADER FILE: FAST_DAT:CAP
SOURCE FILE: YOURFILE.MAC

< THE OUTPUT WILL BE YOURFILE.BIN >

2.2 DCL FOREIGN

IN ADDITION TO USING THE DCL SRUN COMMAND, FASTBIN MAY ALSO BE ACTIVATED BY THE DCL FOREIGN COMMAND LINE SYSTEM. THIS METHOD REQUIRES THAT EITHER THE USER OR PROGRAM INSTALLER HAS SET UP THE SYMBOL:

SPASTBIN == SPAST_EXE:

WITH THIS SYMBOL SET UP, THE FASTBIN USER MAY CALL FASTBIN USING THE COMMAND:

SPASTBIN [SOURCE FILE]

WHERE [] DENOTES AN OPTIONAL PARAMETER.

FASTBIN WILL PROMPT FOR:

1. 'HEADER FILE:' SEE DESCRIPTION IN SECTION 2.1. AND TABLE 1.

IF THE USER DID NOT SPECIFY <SOURCE_FILE> IN THE FASTBIN COMMAND THEN FASTBIN WILL PROMPT FOR:

2. 'SOURCE FILE:' SEE DESCRIPTION IN SECT 2.1.

WHEN ACTIVATED VIA THIS FOREIGN COMMAND METHOD, FASTBIN ALWAYS PRODUCES AN OUTPUT FILE WITH THE SAME NAME AS THE SOURCE FILE BUT WITH A FILE TYPE OF .BIN. THE ADVANTAGE OF DCL FOREIGN OVER DCL SRUN COMES WHEN USING CERTAIN PROCEDURES. DCL FOREIGN CAN RELY ON SYMBOL SUBSTITUTION FOR THE <SOURCE FILE> WHILE DCL SRUN CANNOT.

EXAMPLE:

SPASTBIN YOURFILE.MAC
HEADER FILE: CAP

< THE OUTPUT WILL BE YOURFILE.BIN >

2.3 EXTENDED DCL SPASTBIN

THE FINAL MANNER IN WHICH FASTBIN MAY BE ACTIVATED IS THROUGH THE EXTENDED DCL COMMAND SYSTEM. THIS METHOD REQUIRES THAT THE FASTBIN INSTALLER SET UP FASTBIN IN THE EXTENDED COMMAND FORM DESCRIBED IN APPENDIX C. WHEN THIS CONDITION IS SATISFIED, THE USER MAY INVOKES FASTBIN USING THE DCL SYNTAX SHOWN IN APPENDIX A, JUST LIKE ANY OTHER DCL COMMAND.

NOTE

SHOULD THE FASTBIN IMAGE BE SET UP TO BE RUN FROM BOTH DCL FOREIGN (SEE SECT 2.2 AND APPENDIX C) AND EXTENDED DCL, THE FOREIGN COMMAND METHOD WILL TAKE PRECEDENCE.

3.0 THE FASTBIN OUTPUT BINARY FILE

SHORTLY AFTER ISSUING A LEGAL FASTBIN COMMAND, FASTBIN WILL TERMINATE LEAVING A BINARY LOAD MODULE FILE IN THE SPECIFIED OR DEFAULT DIRECTORY. THIS OUTPUT FILE HAS THE SAME NAME AS THE SOURCE FILE WITH A FILE TYPE OF .BIN UNLESS THIS DEFAULT WAS OVERRIDDEN.

THE OUTPUT FILE CAN NOW BE SPECIFIED AS INPUT TO THE PSPUR LOADER UTILITY WHICH WILL LOAD THE BINARY OBJECT CODE INTO THE APPROPRIATE PSPX+ PROCESSOR. THE CODE IS THEN READY FOR EXECUTION.

4.0 CUSTOM FASTBIN HEADERS

IN ADDITION TO THE STANDARD PSPX+ COMPONENT HEADERS MENTIONED IN TABLE 1, IT IS POSSIBLE FOR A USER TO GENERATE A CUSTOM HEADER FILE THAT IS ACCEPTABLE TO FASTBIN AND THAT MAY BE USED INSTEAD OF THE STANDARD HEADERS LISTED IN THE TABLE. THIS IS DONE USING THE WESTINGHOUSE BINGEN UTILITY:

1. CREATE AN EMPTY SOURCE FILE IN THE FOLLOWING MANNER:

```
SCREATE EMPTY.MAC
```

```
^Z
```

```
S
```

WHERE Z IS CONTROL Z.

2. CALL BINGEN AND SPECIFY THE PSPX+ COMPONENT PROCESSOR AND PARAMETERS THAT YOU WISH THE GENERATED HEADER FILE TO HAVE.
3. WHEN BINGEN PROMPTS FOR THE SOURCE FILE NAME, SPECIFY EMPTY.MAC.
4. WHEN BINGEN PROMPTS FOR THE OUTPUT FILE NAME, SPECIFY XXXX.CAT WHERE XXXX STANDS FOR WHATEVER IS MEANINGFUL TO YOU.

FASTBIN USER'S MANUAL AND INSTALLATION GUIDE
CUSTOM FASTBIN HEADERS

PAGE 6
17 JAN 83

THE XXXX.DAT FILE IS THE NEW CUSTOM HEADER WHICH MAY BE
SPECIFIED AS INPUT TO FASTBIN. IN ESSENCE A HEADER FILE IS
THE LOAD MODULE FOR THE EMPTY PROGRAM AND THAT IS EXACTLY
WHAT THESE STEPS HAVE GENERATED.

APPENDIX A

EXTENDED DCL FASTBIN COMMAND SYNTAX

A.1 SFASTBIN

INVOKES THE FAST PSPX+ BINARY LOAD MODULE GENERATOR WHICH CREATES FILES ACCEPTABLE AS INPUT TO THE PSP LOADER UTILITY (PSPLDR).

A.1.1 FORMAT

SFASTBIN <FILE_SPEC>

QUALIFIERS -----	DEFAULT -----
/CAP	NONE. (1)
/IUC1	"
/IUC2	"
/VAP1	"
/VAP2	"
/BULKMEM	"
/MXM	"
/EXPERIMENTAL=<FILE>	SEE TEXT.
/OUTPUT=<BIN_SPEC>	SEE TEXT.

(1). IF /EXPERIMENTAL IS NOT USED THEN ONE OF THE FIRST SEVEN MUST BE SPECIFIED OR ELSE FASTBIN WILL PROMPT 'HEADER FILE:'. FASTBIN WILL PROMPT 'HEADER FILE:'. SEE SECTION 2.1 AND TABLE 1.

EXTENDED DCL FASTBIN COMMAND SYNTAX
SFASTBINPAGE A-2
17 JAN 83

A.1.2 PROMPTS

FILE: <FILE_SPEC>
HEADER FILE: <SEE SECTION 2.1 AND TABLE 1>

A.1.3 COMMAND PARAMETERS

1. <FILE_SPEC> SPECIFIES AN ASCII OCTAL OBJECT FILE THAT WILL BE USED TO CREATE A PSPX+ BINARY LOAD MODULE. IF YOU DO NOT SPECIFY A FILE TYPE, .MAC IS USED BY DEFAULT. THIS PARAMETER IS REQUIRED AND NO WILDCARDS ARE ALLOWED IN THE FILE_SPEC.

A.1.4 QUALIFIERS

1. /CAP, /IUC1, /IUC2, /VAP1, /VAP2, /BULKMEM, /AXM CONTROL THE TYPE OF LOAD MODULE THAT WILL BE CREATED. A UNIQUE LOAD MODULE TAILORED TO A SPECIFIC PSPX+ COMPONENT PROCESSOR WILL BE CREATED BASED ON THIS QUALIFIER. ONE OF THESE MUST BE SPECIFIED UNLESS /EXPERIMENTAL IS USED IN WHICH CASE NONE OF THESE ARE ALLOWED.
2. /OUTPUT = <BIN_SPEC> CONTROLS THE NAME OF THE OUTPUT BINARY LOAD MODULE FILE. BY DEFAULT FASTBIN PRODUCES AN OUTPUT FILE THAT HAS THE SAME NAME AS THE INPUT SOURCE FILE EXCEPT WITH A FILE TYPE OF .BIN. WHEN YOU SPECIFY /OUTPUT YOU CAN OVERRIDE THIS DEFAULT.
3. /EXPERIMENTAL = <FILE> IS USUALLY USED FOR DEVELOPEMENT ONLY. THIS QUALIFIER ALLOWS THE USER TO SPECIFY A LOAD MODULE HEADER OTHER THAN THOSE SELECTED BY THE /CAP, /IUC1, /VAP1 ETC... QUALIFIERS. THIS EXPERIMENTAL HEADER MUST BE CREATED USING THE BINGER COMMAND PROCEDURE. /EXPERIMENTAL MAY NOT BE USED WITH /CAP, /IUC1, /VAP ETC. QUALIFIERS.

EXTENDED DCL FASTBIN COMMAND SYNTAX
SFASTBIN

PAGE A-3
17 JAN 83

A.1.5 EXAMPLES

1. SFASTBIN/CAP TESTFILE
THIS COMMAND GENERATES A CAP LOAD MODULE CALLED
TESTFILE.BIN FROM THE INPUT FILE TESTFILE.CAP .
2. SFASTBIN/VAPI/OUTPUT=ZZZ.ABC TST.XXX
THIS COMMAND GENERATES A VAPI LOAD MODULE CALLED
ZZZ.ABC FROM THE SOURCE FILE TST.XXX .
3. SFASTBIN/EXPERIMENTAL=BMM.DAT TESTFILE
THIS COMMAND GENERATES A BINARY LOAD MODULE CALLED
TESTFILE.BIN FROM THE SOURCE FILE TESTFILE.CAP .
THE EXPERIMENTAL HEADER FILE BMM.DAT DESCRIBES THE
PSPX+ COMPONENT PROCESSOR FOR WHICH TESTFILE.BIN
WAS GENERATED.

NOTE

SFASTBIN/EXPERIMENTAL=FAST_DAT:VAPI.DAT TESTFILE
PRODUCES THE SAME RESULTS AS
SFASTBIN/VAPI TESTFILE
THE SAME APPLIES TO CAP.DAT, IOC1.DAT, VAPI.DAT
ETC.

APPENDIX B

FASTBIN INSTALLATION GUIDE

B.1 INTRODUCTION

THIS APPENDIX DESCRIBES HOW TO RECOVER THE FASTBIN FILES FROM TAPE AND INSTALL THE FASTBIN.EXE IMAGE ON A VAX SYSTEM THAT IS ALREADY EQUIPPED WITH A WESTINGHOUSE PSP SUPPORT SOFTWARE PACKAGE.

B.2 BACKUP TAPE

THE FASTBIN SOFTWARE IS PROVIDED ON A MAGNETIC TAPE IN BACKUP FORMAT. THE BACKUP UTILITY IS DESCRIBED IN THE VAX/VMS UTILITIES REFERENCE MANUAL. TO RECOVER THE FILES, MOUNT THE TAPE FOREIGN (SMOUNT/FOREIGN) AND RECOVER THE FILES INTO YOUR FAVORITE DIRECTORY THUS:

```
SBACKUP/VERIFY MTA0: <YOUR DIRECTORY SPECIFICATION>
```

B.3 PLACING THE FASTBIN IMAGE

THIS SECTION DESCRIBES WHERE TO PLACE THE REQUIRED FASTBIN FILES.

1. COPY THE FOLLOWING FILES INTO THE SYSTEM PSPSSYSTEM: DIRECTORY.

```
FASTBIN.EXE  
FASTBIN.CLD  
CAP.DAT  
IOC1.DAT  
VAP1.DAT  
VAP2.DAT  
SM.DAT  
MXM.DAT
```

FASTBIN INSTALLATION GUIDE
PLACING THE FASTBIN IMAGEPAGE B-2
17 JAN 83

-
2. EDIT THE SYSTEM STARTUP FILE
SYSSMANAGER:SYSTARTUP.COM AND ADD THE FOLLOWING
COMMANDS:
-

SASSIGN/SYSTEM PSP\$SYSTEM: FAST_DAT:
SASSIGN/SYSTEM PSP\$SYSTEM:FASTBIN.EXE FAST_EXE:

B.4 CHOOSING THE ACTIVATION METHOD

AT THIS POINT YOU MUST DECIDE HOW YOU WANT USERS TO ACTIVATE
THE FASTBIN.EXE IMAGE. THE THREE OPTIONS ARE DESCRIBED IN
SECTION 2 OF THE USER'S MANUAL. WHEN IN DOUBT IT IS
SUGGESTED YOU USE DCL FOREIGN. THE FOLLOWING SECTIONS
DESCRIBE WHAT YOU MUST DO DEPENDING ON HOW YOU WANT TO
ACTIVATE FASTBIN.

B.4.1 DCL SRUN ACTIVATION

IF YOU CHOOSE TO ACTIVATE FASTBIN BY DCL SRUN, THE
INSTALLATION IS COMPLETE. USERS MAY CALL FASTBIN BY:

SRUN FAST_DAT:FASTBIN

B.4.2 DCL FOREIGN

IF YOU CHOOSE TO ACTIVATE FASTBIN VIA THE DCL FOREIGN
COMMAND SYSTEM THEN FOLLOW THESE STEPS:

1. EDIT THE SYSTEM LOGIN FILE SYSSMANAGER:SYLOGIN.COM
AND ADD THE COMMAND
SFAST*BIN ::= SFAST_EXE:
-

THE INSTALLATION IS NOW COMPLETE. FASTBIN MAY BE RUN IN
ACCORDANCE WITH SECTION 2.2 OF THE USERS MANUAL.

B.4.3 EXTENDED DCL SFASTBIN

IF YOU CHOOSE TO USE EXTENDED DCL THEN FOLLOW THESE STEPS:

1. EDIT THE SYSTEM LOGIN FILE SYSSMANAGER:SYLOGIN.COM
AND ADD THE COMMAND
SSET COMMAND FAST_DAT:FASTBIN.CLD
-

FASTBIN INSTALLATION GUIDE
CHOOSING THE ACTIVATION METHOD

PAGE 8-3
17 JAN 83

2. EDIT THE SYSTEM STARTUP FILE
SYSSMANAGER:SYSTARTUP.COM AND ADD THE COMMAND
SMCR INSTALL SYSSSYSTEM:CLEDITOR/PRIV=(CMEXEC)

THE INSTALLATION IS NOW COMPLETE. FASTBIN MAY BE RUN IN
ACCORDANCE WITH SECTION 2.3 OF THE USER'S MANUAL.

AD-A130 655

MULTIMODE RADAR SIGNAL PROCESSOR INTEGRATION FACILITY
(U) AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON
AFB OH J N HORN ET AL. MAY 83 AFWAL-TR-83-1045

2/2

UNCLASSIFIED

F/G 17/9

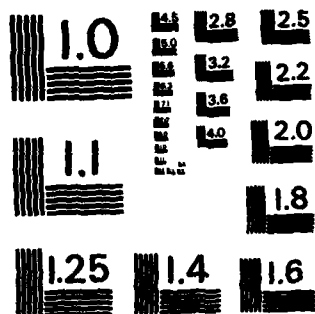
NL



END

DATE
FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX C
CAS COMMAND FILE

```

S! PROGRAM CAS----THIS PROGRAM INVOKES THE NECESSARY FILE MANIPULATIONS
S! FOR ASSEMBLY OF A CAP PROGRAM. IT ALSO INVOKES TRSCAP AND FASTBIN,
S! IF REQUESTED. PRINTING OF LIST FILES IS AVAILABLE TO THE USER.
S! SEVERAL QUALIFIERS ARE AVAILABLE TO EXECUTE A PART OF THE ASSEMBLY
S! PROCESS OR ADD THE DEASSEMBLY OPTION.
S!
S! TO RUN THE PROGRAM TYPE: CAS OPTIONS
S!
S! AVAILABLE OPTIONS ARE:
S!
S!      I----INITIALIZE THE REL FILE AND EXIT.
S!      A----ASSEMBLE THE PROGRAM.
S!      L----LOAD THE PROGRAM AND EXIT.
S!      LT---LOAD AND TRSCAP THE PROGRAM.
S!      LTF---LOAD, TRSCAP AND FASTBIN THE PROGRAM.
S!      D----ASSEMBLE, LOAD AND DEASSEMBLE THE PROGRAM.
S!      DT---ASSEMBLE, LOAD, DEASSEMBLE AND TRSCAP THE PROGRAM.
S!      DTF---ASSEMBLE, LOAD, DEASSEMBLE, TRSCAP AND FASTBIN THE PROGRAM.
S!      DL---LOAD AND DEASSEMBLE THE PROGRAM.
S!      DLT---LOAD, DEASSEMBLE AND TRSCAP THE PROGRAM.
S!      DLTf---LOAD, DEASSEMBLE, TRSCAP AND FASTBIN THE PROGRAM.
S!      NO OPTION(DEFAULT)----ASSEMBLE, LOAD, TRSCAP AND FASTBIN PROGRAM.
S!
S! WRITTEN BY B.R. STEPHENS.      BECAME OPERATIONAL 16-JUL-82.
S!
S! MODIFIED 8-OCT-82 TO INCLUDE THE FASTBIN OPTION.
S!
S! THIS PROGRAM INVOKES SSP AND TRSCAP.
S! NOTE: ENTER ONLY THE FILENAME WHEN REQUESTED, NO EXTENSION!!
S
S INQUIRE FILENAME ENTER THE FILENAME      ! GET THE FILENAME
S IF 'F$LENGTH(FILENAME)' .GT. 9 THEN GOTO BADFIL
S ASSIGN 'FILENAME'.REL FOR050
S IF P1.EQS."L" .OR. P1.EQS."LT" THEN GOTO LOAD1      ! IF NOT ASS THEN SKIP
S IF P1.EQS."DL" .OR. P1.EQS."DLT" THEN GOTO LOAD1
S IF P1.EQS."LTF" .OR. P1.EQS."DTf" THEN GOTO LOAD1
S CREATE CAS.DAT                                ! CREATE INITIALIZATION FILE
S FILE
ASG,U 50
S STOP
S
S ASSIGN/USER_MODE NL: SYSSOUTPUT
S SSP CAS.DAT                                ! INITIALIZE FILE
S DELETE CAS.DAT;*,FOR026;*
S IF P1.EQS."I" THEN EXIT                      ! IF ONLY INITIALIZING THEN EXIT
S ASSIGN 'FILENAME'.LIS FOR006                ! ASSEMBLY LIST FILE
S SSP 'FILENAME'.CAP                          ! ASSEMBLE THE PROGRAM
S IF P1.EQS."A" THEN GOTO ENDA                ! IF ONLY ASSEMBLING THEN EXIT
S LOAD1:
S ASSIGN 'FILENAME'.ADS FOR027
S OPEN/WRITE F FM.DAT                        ! CREATE LOAD FILE
S WRITE F "; FILE"
S WRITE F "ASG,A 50"
S WRITE F ";A LOAD"
S WRITE F "IN 50," FILENAME
S WRITE F "LIS 32"
S WRITE F "END"
S DET:=F$EXTRACT(0,1,P1)
S IF DET.EQS."D" THEN GOTO DEAS              ! IF DEASSEMBLING SET UP LIST FILE
S DEASSIGN FOR006
S ASSIGN/USER_MODE NL: SYSSOUTPUT

```

Copy available to DTIC does not
 permit fully legible reproduction

```

S GOTO LOAD
S DEAS:
S WRITE F ";P* DEASSEMBLE"
S ASSIGN 'FILENAME'.LST FOR006
S LOAD:
S WRITE F "; STOP"
S CLOSE F
S SSP FM.DAT ! INVOKE THE LOADER
S IF DET.EQS. "D" THEN GOTO PD
S! IF STARTED WITH LOAD, SKIP ASSEMBLY LIST REQUEST
S IF P1.EQS."L" .OR. P1.EQS."LT" .OR. P1.EQS."LTF" THEN GOTO SKIPAS
S ENDA:
S INQUIRE ANS DO YOU WANT AN ASSEMBLY LISTING?
S IF ANS THEN PRINT 'FILENAME'.LIS
S PURGE 'FILENAME'.* ! PURGE AND DELETE FILES AS NECESSARY
S SKIPAS:
S DELETE FOR028.DAT;* ,FOR026;* ,FOR025;* , 'FILENAME'.REL;*
S DEASSIGN FOR050
S IF P1.EQS. "A" THEN GOTO NOTA
S CONT:
S! IF NOT INVOKING TRSCAP THEN EXIT
S IF P1.EQS."A" .OR. P1.EQS."D" THEN EXIT
S IF P1.EQS."L" .OR. P1.EQS."DL" THEN EXIT
S OPEN/WRITE F FM.DAT ! CREATE TRSCAP FILE
S WRITE F FILENAME,".ABS"
S WRITE F FILENAME,".MAC"
S CLOSE F
S ASSIGN FM.DAT FOR005
S ASSIGN/USER_MODE NL: SYSSOUTPUT
S TRSCAP ! INVOKE TRSCAP
S DEASSIGN FOR005
S DELETE FM.DAT;* , 'FILENAME'.ABS;*
S! IF NOT USING FASTBIN, THEN EXIT.
S IF P1.EQS."LT" .OR. P1.EQS."DT" .OR. P1.EQS."DLT" THEN EXIT
S FAST 'FILENAME' ! FASTBIN THE PROGRAM.
CAP
S DELETE 'FILENAME'.MAC;*
S PURGE 'FILENAME'.BIN
S EXIT
S BADFIL:
S WRITE SYSSOUTPUT "FILENAME IS TOO LONG!"
S EXIT
S PD: ! ASK IF DEASSEMBLY LISTING IS NEEDED
S INQUIRE ANS DO YOU WANT A DEASSEMBLY LISTING?
S IF ANS THEN PRINT 'FILENAME'.LST
S DEASSIGN FOR006
S DELETE FOR034.DAT;*
S! IF STARTED WITH LOAD SKIP ASSEMBLY LIST REQUEST
S IF P1.EQS."DL" .OR. P1.EQS."DLT" THEN GOTO SKIPAS
S GOTO ENDA
S NOTA:
S DEASSIGN FOR027
S DELETE FM.DAT;* ,FOR033;*
S GOTO CONT

```

REFERENCES

1. Rafael Gonzalez and Paul Wintz, Digital Image Processing, Addison-Wesley Publishing Company, 1977.
2. PSP-X+ IOC/VAP/CHANNEL BUS USERS MANUAL, 9RA8001H01, Westinghouse Defense and Electronics Systems Center, January 1982.
3. PSP-X+ CAP PROGRAMMERS MANUALS, 9RA8002H01, Westinghouse Defense and Electronics System Center, January 1982.
4. PSP-X+ COMPILERS REFERENCE MANUAL, 9RA6843H02, Westinghouse Defense and Electronics Systems Center, January 1982.
5. RAMTEK RM-9050 PROGRAMMING MANUAL, Part No. 503746C, February 1979.
6. VAX/VMS VOLUME 2B, Guide to Using Command Procedures, Version 3.0.