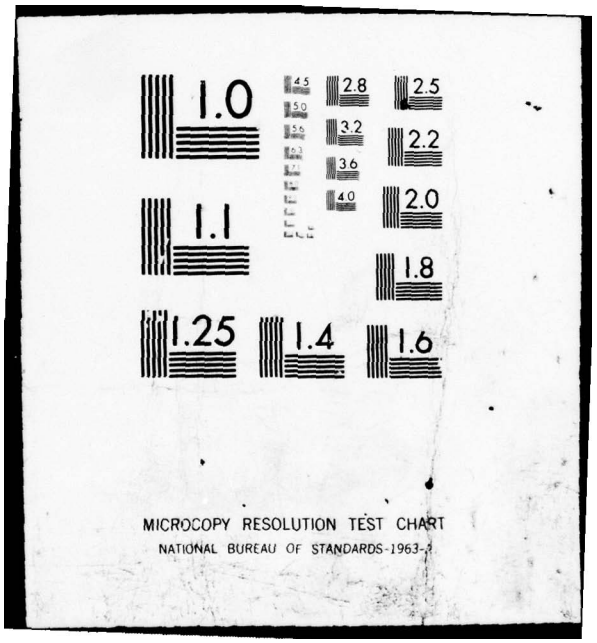
AD-A067 693 RAYTHEON CO SUDBURY MASS EQUIPMENT DIV F/G 9/2
FAULT TOLERANT SPACEBORNE COMPUTER STUDY. PREPROCESSOR DESIGN.(U)
JAN 79 J J STIFFLER F04701-77-C-0050
UNCLASSIFIED ER79-4005 SAMSO-TR-79-27 NL

1 OF 2
AD
A067693


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SAMSO—TR—79—27

LEVEL

12

FAULT TOLERANT SPACEBORNE COMPUTER STUDY

RAYTHEON COMPANY
SUDBURY, MASSACHUSETTS 01776

9 JANUARY 1979

PREPROCESSOR DESIGN REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

PREPARED FOR

SPACE AND MISSILE SYSTEMS ORGANIZATION
AIR FORCE SYSTEMS COMMAND
LOS ANGELES AIR FORCE STATION
BOX 92960, WPC
LOS ANGELES, CALIFORNIA 90009

RAYTHEON COMPANY
EQUIPMENT DIVISION

FAULT-TOLERANT SPACEBORNE COMPUTER (FTSC) STUDY

CONTRACT #: F04701-77-C0050 new

PREPROCESSOR DESIGN

CDRL 003A2

ER79-4005

9 January 1979

RAYTHEON COMPANY
EQUIPMENT DIVISION
EQUIPMENT DEVELOPMENT LABORATORIES
SUDBURY, MASSACHUSETTS

This final report was submitted by Raytheon Co., Equipment Division, Strategic Systems Directorate, Sudbury, Massachusetts, under contract F04701-77-C-0050 with the Space and Missile Systems Organization, Los Angeles Air Force Station, Los Angeles, California. Second Lieutenant Douglas D. Orville (YCD) is the Project Officer.

This report has been reviewed by the Information Office (OI) 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. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

Douglas D. Orville

DOUGLAS D. ORVILLE, 2LT, USAF
Project Officer
Advanced Space Computer Division

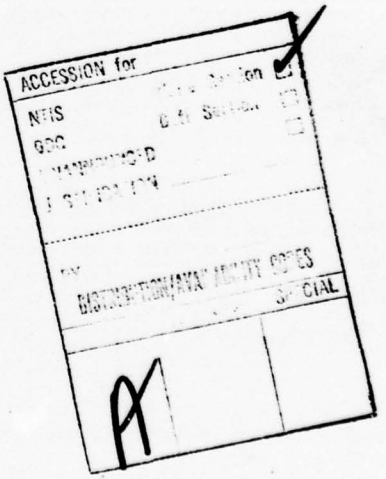
Ronald G. Spray

RONALD G. SPRAY, LT COL, USAF
Deputy Director for Subsystems

FOR THE COMMANDER

Leonard E. Baltzell

LEONARD E. BALTZELL, Col, USAF
Asst. Deputy for Advanced
Space Programs



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: (18) SAMSO (19) TR-79-27
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: CDRL 003A2

4. TITLE *(and Subtitle)*: (6) FAULT TOLERANT SPACEBORNE COMPUTER STUDY. Preprocessor Design.
5. TYPE OF REPORT & PERIOD COVERED: (9) Final Technical Report. 1 Jun 77 - 28 Feb 79.
6. PERFORMING ORG. REPORT NUMBER: (14) ER79-4005

7. AUTHOR(s): (10) J.J. Stiffler
8. CONTRACT OR GRANT NUMBER(s): (15) F04701-77-C-0050

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Raytheon Co.
Equipment Division, Strategic Systems Dir..
Sudbury, MASS 01776

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
PE63401F (16)
Project 2181

11. CONTROLLING OFFICE NAME AND ADDRESS
Space and Missile Systems Orgn. (YCD)
P.O. Box 92960 Worldway Postal Center
Los Angeles, CA 90009

12. REPORT DATE: (11) 9 Jan 79
13. NUMBER OF PAGES: 158

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: (12) 158 p.

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
SAMSO Project Officer: 2LT Douglas D. Orville (YCD)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Wide Band Signal Processor
Fault-Tolerant Spaceborne Computer
Interleaver/De-Interleaver Processor
Demodulator Processor
Decoder Processor

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
This report defines an architecture for a highly reliable spaceborne communications processor. The processor was to be designed to perform the demodulation, decoding, de-interleaving, interleaving, encoding, formatting and routine tasks anticipated for the Air Force Nuclear Forces Communications Satellite. The goal was to define a system having a long-term reliability comparable to that of the SAMSO Fault-Tolerant Spaceborne Computer and having a comparably low redundancy overhead. This technical report will be updated in the future to present a more detailed picture of the Wide Band Signal Processor.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

405 037

Table of Contents

TABLE OF CONTENTS (CONT'D.)

TABLE OF CONTENTS (CONT'D.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (CONT'D.)

LIST OF TABLES

WIDE-BAND SIGNAL PROCESSOR

1.0 INTRODUCTION

The purpose of the Wide-Band Signal Processor (WBSP) study was to define an architecture for a highly reliable spaceborne communications processor. The processor was to be designed to perform the demodulation, decoding, de-interleaving, interleaving, encoding, formatting and routing tasks anticipated for the SSS satellites. The goal was to define a system having a long-term reliability comparable to that of the Fault-Tolerant Spaceborne Computer (FTSC) and having a comparably low redundancy overhead. This report summarizes the results of that study.

The FTSC itself served as the starting point for the study for two reasons:

- Considerable effort had gone into the FTSC design to make it truly fault tolerant and, in particular, to ensure that faults could be isolated and the faulty module replaced. The ability to isolate faults is crucial to any fault-tolerant design; it was clearly advantageous, therefore, to appropriate for the WBSP all the applicable fault-tolerant features of the FTSC.
- Roughly twenty new, radiation hardened, CMOS/SOS LSI devices are being developed for the FTSC. The WBSP development cost could be significantly reduced by designing it to be implemented to the greatest extent possible with these FTSC devices.

The WBSP architecture resulting from this effort is described in the following section. It is designed to operate as a peripheral to the FTSC (cf, Figure 1-1).

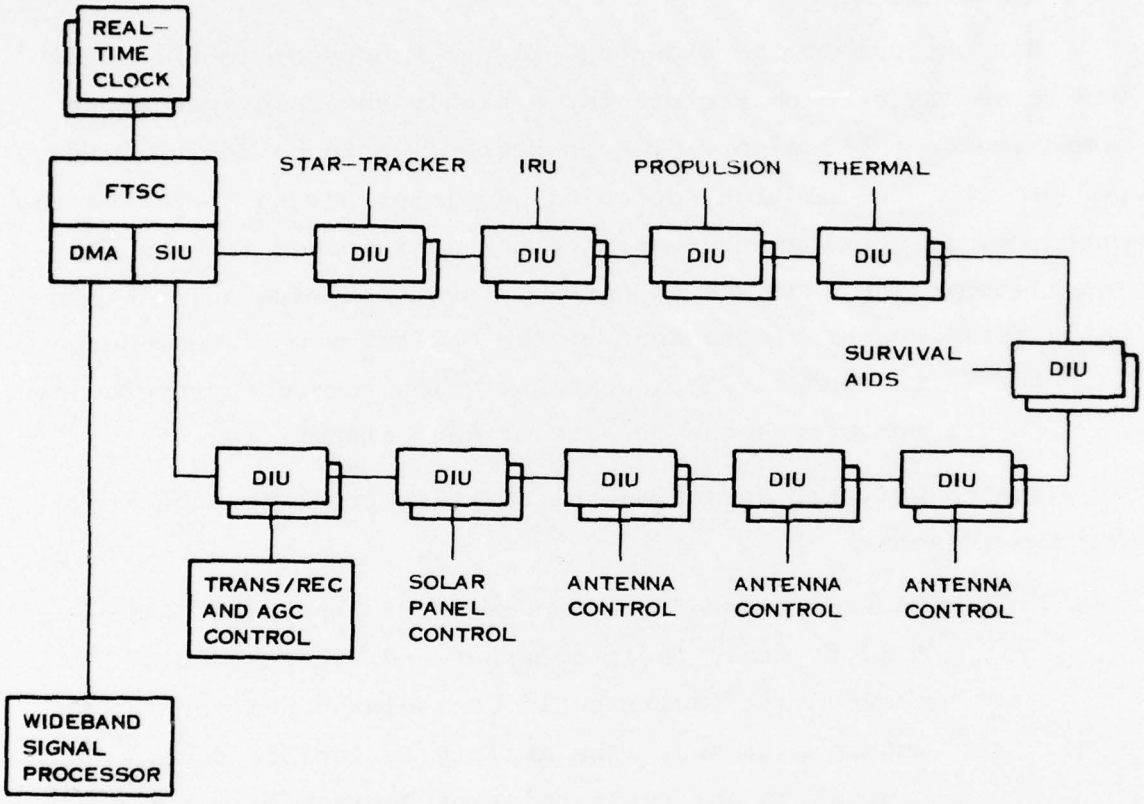


FIGURE 1-1

SSS PROCESSOR CONFIGURATION

Three types of processors are used in the WBSP: a core 8-bit microprocessor and two variations. The core processor is described in Section 3 along with the various microinstruction algorithms that have been developed for it. The two variations, the demodulator processor and the de-interleaver processor, are discussed in Sections 4 and 5, respectively. Reliability, power and weight estimates for the WBSP and the FTSC are presented in Section 6 along with an assessment of the magnitude of the processing tasks (navigation, guidance, control and general housekeeping tasks as well as communications tasks) relegated to the FTSC. Section 7 concludes with an assessment of the results to date and recommendations for further work.

2.0 WBSP SYSTEM CONFIGURATION

One of the most important lessons learned during the early phases of the FTSC study concerned the dramatic improvement in reliability that could be achieved by pooling spares. A spare module capable of replacing any one of, say, five active modules in the event of a failure improves the system reliability nearly as much as if each of the five modules had a dedicated spare; yet the latter configuration obviously requires five times as much redundant hardware. Two conditions must be satisfied, however, in order to exploit this potential:

- The switching device used to isolate the faulty module and to replace it with one of the pooled spares must be so designed that its unreliability does not dissipate much of the potential of this approach.
- The processor must be partitioned to the largest extent possible into interchangeable modules so that spares can in fact be pooled.

The first of these requirements was addressed extensively in the FTSC design and the results of this effort were adapted with only minor modifications for the WBSP. The second requirement was relatively easily satisfied in the WBSP due to the fact that it is inherently a multichannel processor; i.e., the processor must perform similar operations on a number of parallel channels. This fact alone suggests a multiprocessor configuration. As will be seen, the WBSP architecture takes full advantage of this inherent parallelism.

The initial candidate WBSP architecture consisted of a number of identical processors and a memory system all attached

to a single bus structure. The active processors were assigned various functions (e.g., demodulation, decoding) under FTSC control. Data were transferred from one processor or memory to another (e.g., from a demodulator to the de-interleaver memory) under the control of a dedicated bus controller. When a fault was detected, the FTSC intervened, isolating the faulty module and programming a spare module to take over its function.

The major disadvantage of the single-bus configuration was that the number of loads on the bus became quite large, causing excessive bus capacitance, and hence limited throughput, combined with high throughput requirements. Several modifications of the original configuration were investigated in an attempt to overcome this problem. The most promising configuration, and the one to be described in the remainder of this report, involves four buses, all controlled by a single (dual-redundant) controller. The four-bus configuration obviously reduces both the number of loads and the throughput requirements on each bus. It does restrict the extent to which spares can be pooled, however, since a module can serve as a spare only for those modules interfacing with the same bus. This disadvantage is compensated for by the fact that the modules associated with each bus can be specifically tailored to a more restricted set of processing tasks, thus increasing their efficiency. The advantages of this feature of the recommended WBSP configuration will become apparent in the following discussion.

The selected WBSP configuration is shown in Figure 2-1. Each of the four buses consists of an eight-bit data byte, an eight-bit address byte, a data parity bit, an address parity bit and a spare byte to be used to replace any failed data or address byte or parity segment. Associated with each bus are two triple-

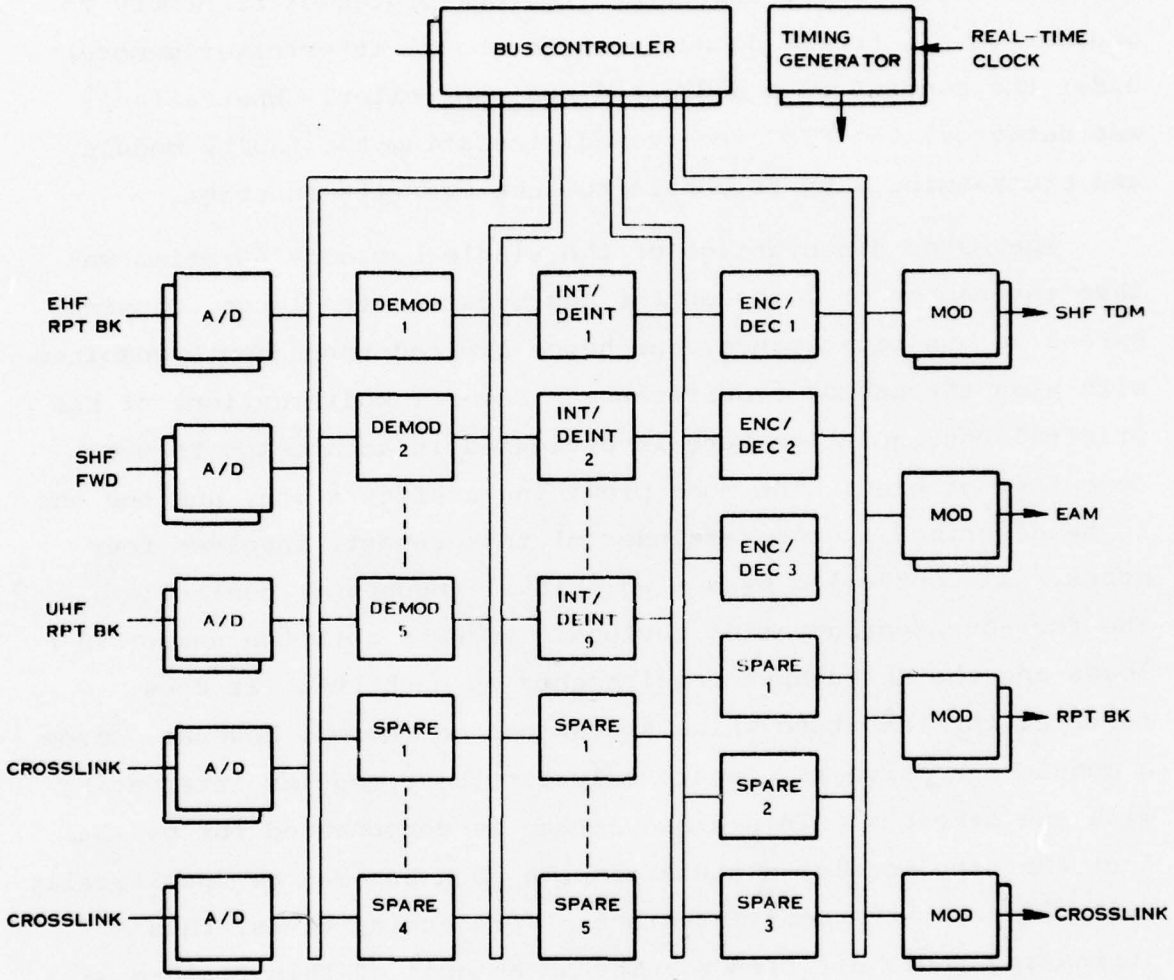


FIGURE 2-1

WIDE-BAND SIGNAL PROCESSOR

modular redundant (TMR) control lines. One is used by the controller to indicate that a valid address is on the address bus; the second is used to distinguish between hard and soft address modes (see below).

In normal operation, the bus controller simply reads from its own memory a sequence of addresses for each of the four address buses, gates these addresses onto the buses and raises the valid-address control signal thereby initiating a new bus cycle. The most significant half of the address designates the address of the module that is to transmit onto the data bus during the following bus cycle; the least significant half designates the module that is to receive the data currently being transmitted.

The addresses used in normal operation are referred to as "soft" addresses. These addresses designate functions rather than modules. That is, a module responds to a given soft address only if it has been previously programmed to implement the function corresponding to that soft address. This initialization is done using "hard" addresses. Each module is permanently assigned a unique (for a given bus) hard, or physical, address to which it responds only in the hard address mode. This mode is used only to test and to reconfigure the WBSP. When a fault is detected either by the FTSC or by the fault monitors (decoders) associated with each module interface, the FTSC is interrupted and, in effect, takes over control of the WBSP. By transmitting hard addresses over the WBSP address buses, it can test any specified module, deactivate it if necessary, activate a spare module and program it (by loading its control memory with the appropriate microinstructions) to take over the function vacated by the failed module. The FTSC then allows the WBSP to resume normal operation.

(The bus structure is critical to the fault tolerance of the system. Provisions must be made, for example, to ensure that failed modules can be deactivated, regardless of the nature of the failure; that once a failed module is deactivated, it cannot disable either of the buses with which it communicates, etc. All of these contingencies were successfully addressed in the FTSC design. Accordingly, except for the modifications needed to accommodate the narrower WBSP buses, the WBSP bus structure is identical to that used in the FTSC and is implemented exclusively with LSI devices being developed for the FTSC.)

In normal operation (cf, Figure 2-1), the bus controller routes information from the analog-to-digital converters to the demodulator processors; from there the demodulated (soft-decision) chip data are routed to the appropriate de-interleaver processor and then to one of the decoder processors. The decoded data is then stored into the FTSC's main memory where each frame is tested for integrity (by checking the message "tails") and duplicates of messages are eliminated. If a tail check indicates a malfunctioning WBSP module, the FTSC initiates the appropriate fault diagnostic routine and, if necessary, reconfigures the WBSP. Otherwise, the processed message is sent back to the WBSP to one of its de-interleaver processors where it is encoded and interleaved. From there, the bus controller routes it to one of the decoder processors where, if appropriate, it is modified using a Queen's code and cover sequence. It is then sent to the appropriate modulator for down-link or cross-link transmission.

As indicated in Figure 2-1, the A/D converters and the modulators are configured in dual-redundant pairs, with each pair dedicated to a particular r.f. link. This configuration precludes the pooling of spares and hence results in a less efficient

utilization of redundancy than would otherwise be possible. Pooling spare A/D converters or modulators would require analog multiplexers between these devices and their analog interfaces. Although this option need not be ruled out, it should be noted that both the A/D converters and the modulators are relatively simple devices; thus, the fact that redundancy is added somewhat inefficiently is not nearly as significant as it would be, for example, for the considerably more complex demodulator or decoder processors.

The communication link between the WBSP's bus controller and the FTSC could utilize either one of the FTSC's direct memory access (DMA) ports or one of its serial bus device interface units (DIUs). If the DMA port is not pre-empted for some other purpose and if the FTSC and WBSP are to be deployed in reasonably close proximity, the DMA interface is preferable since it enables more rapid communication between the two systems and hence shortens the time needed for diagnosis and recovery. Since the DIU port is fast enough for normal operation, however, the only penalty in using it is in a somewhat longer recovery period following a WBSP failure.

The only fault monitoring currently envisioned for the WBSP are the bus decoders provided at each bus interface and the "tail checking" already mentioned in the FTSC. Since the data are highly encoded, it is felt that the tail checks by themselves will provide an adequate means of monitoring the health of the system. The main function of the bus decoders is to help the FTSC isolate faults involving inter-module communication. If additional fault monitoring is found to be desirable, it would be relatively simple to add one processor on each bus and to program it to monitor, on a time shared basis, the performance of each of the other monitors

on the same bus. (It should be noted that the monitoring processor can in general execute considerably simpler algorithms than the processor being monitored and still thoroughly check the latter's performance.)

The bulk of the processing capability, and hence of the system complexity, in the WBSP resides in the three sets of processors (demodulator processors, interleaver/de-interleaver processors, and encoder/decoder processors) shown in Figure 2-1. Each of these three processor types is discussed in turn in the following three sections.

3.0 THE CORE PROCESSOR

The three types of processors used in the WBSP consist of a core processor (the encoder/decoder processor) and two modifications of that processor (the demodulator and interleaver/de-interleaver processors). The core processor is discussed in this section and the other two processors in the following two sections. Paragraph 3.1 describes the architecture of the core processor. The following paragraphs describe four different signal processing applications for such a processor: a Viterbi decoder metric processor; a Viterbi decoder path processor; a dual-3 decoder and a feedback decoder. The specific function to be implemented by the processor is determined through an initialization phase during which its control RAM is loaded with the appropriate micro-instructions. The processor then accepts information received over the bus, processes that information, stores the result in its output buffer and waits for a new input before repeating the same sequence. The processor is interrupted each time the input buffer receives a new word of information. Since the entire system uses a common clock, this interrupt mechanism is sufficient to keep the system in synchronism.

3.1 GENERAL DESCRIPTION

The core processor consists of an interface section, a control section and a processing section. The block diagram shown in Figure 3-1 shows the LSI devices which comprise these sections. The interface section of the core processor consists of 6 LSI chips of three types, including two Control Interface chips. The control section consists of the I/O Buffer and Control RAM Sequencer and the Control RAM. The Control RAM requires 256 40-bit words, and is implemented using ten 256-word by 4-bit RAMs. The processing section consists of an 8-bit

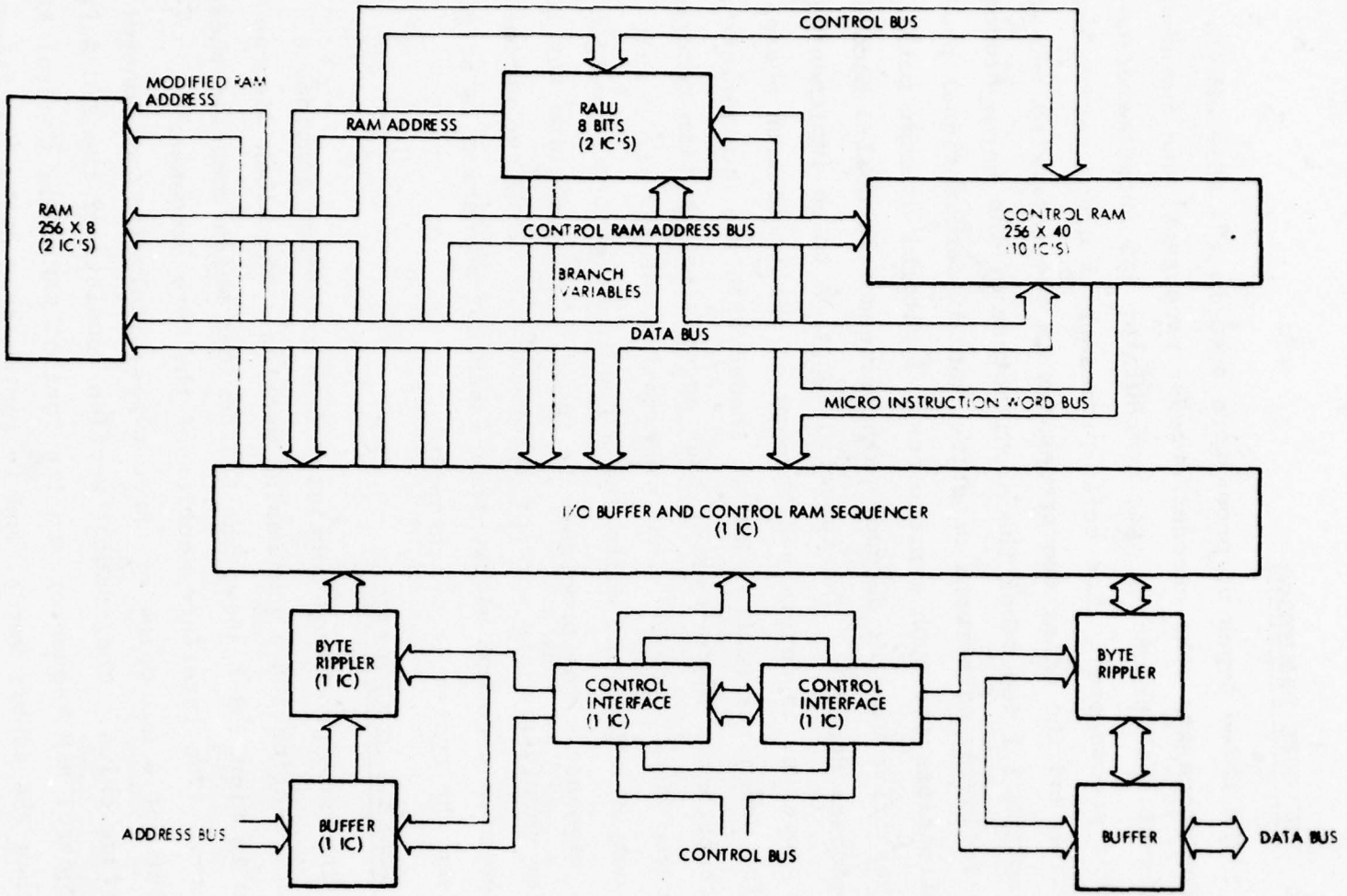


Figure 3-1 GENERAL PURPOSE PROCESSOR

Register Array and Arithmetic Logic Unit (RALU) and a data storage RAM with a capacity of 256 8-bit words. The RALU is implemented with two 4-bit RALU chips and the storage RAM with two 256-word by 4-bit RAM chips. This results in a core processor total chip count of 21 devices.

The three types of interface chips and the RALU are identical to those being developed for the FTSC. The 1024-word by 1-bit RAM being developed by the FTSC program will be designed so that it can be easily changed to a 256-word by 4-bit configuration for the core. The 4-bit FTSC RALU will be directly transferrable to the basic processor without modification. The only LSI device to be developed specifically for the core processor is the I/O Buffer and Control RAM Sequencer.

This latter device provides control to the processor and interfaces the processor to the bus system. It buffers data bytes for transfer between the processor register array and the external bus system. When commanded to load the microprogram control RAM (by a hard-address command), the I/O Buffer and Control RAM Sequencer takes over control of the processor. It sequences through the RAM loading 8 bits at a time from data routed to the input buffer by the bus controller. After the control RAM is loaded, the processor is switched back to normal (again by hard-address command) and begins processing starting at address zero of the control RAM.

An assignment of the control ROM bits is listed in Table 3-1. The block diagram shown in Figure 3-2 indicates the RALU control assignment. The width of the register array input multiplexer field has been reduced from the 6 bits used in the FTSC to 2 bits, by having both the general purpose and working register multiplexers share the same 2 bits.

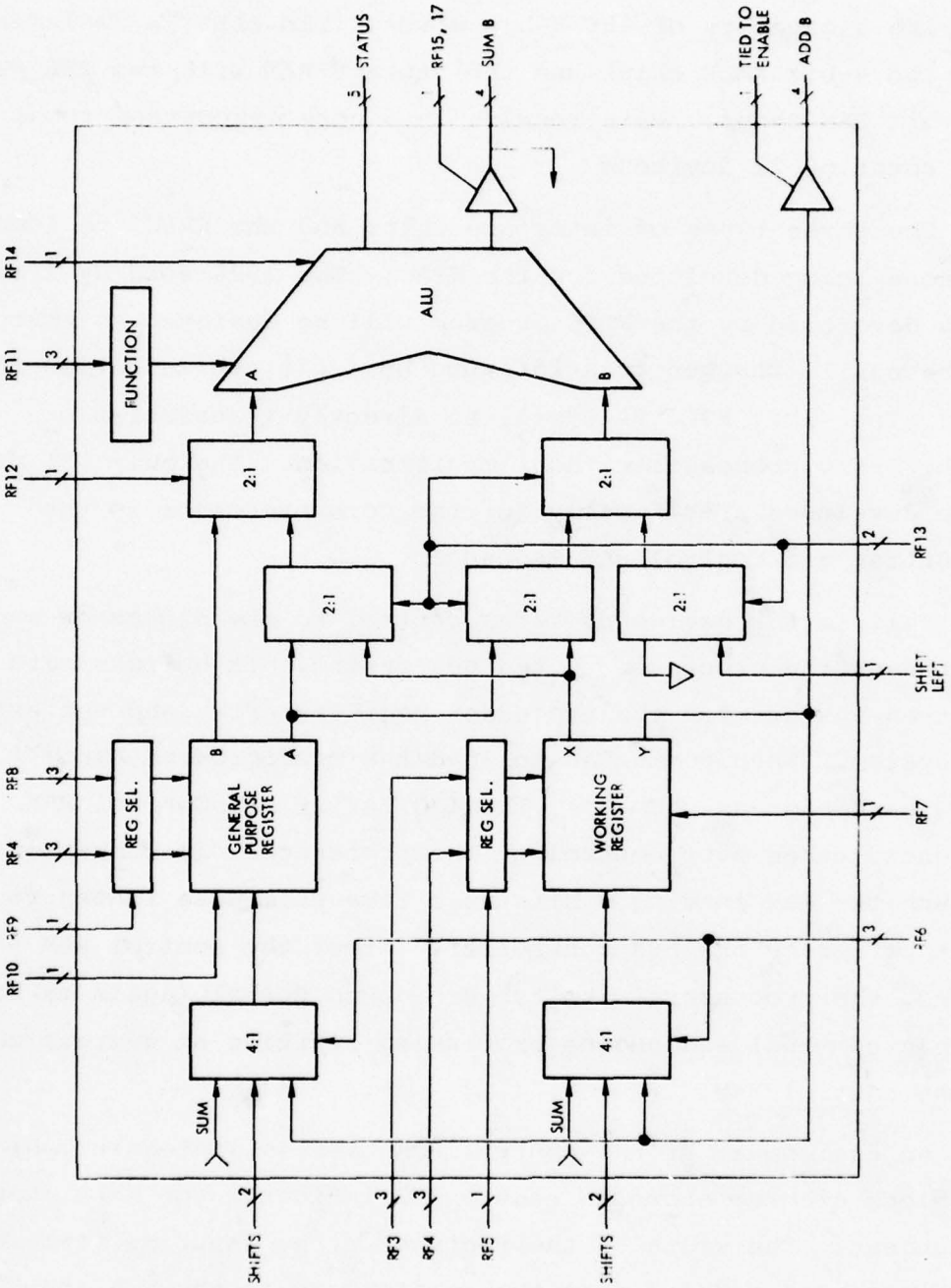


Figure 3-2 WBSP ALU CONFIGURATION

Table 3-1
CONTROL RAM OUTPUT SIGNAL ASSIGNMENTS

RAM Field	Number of Bits	Function
RF01	3	Control RAM sequencer branch address (Up to 7 different branch condition codes may be specified).
RF02	8	Control RAM next address
RF03	3	Working register WX output select
RF04	3	Working register WY output select and General Purpose register RA output select
RF05	1	Working register WX/WY input select
RF06	2	Working register multiplexer input select; General Purpose register multiplexer input select
RF07	1	Working register write clock
RF08	3	General purpose register RB output select
RF09	1	General purpose register RA/RB input select
RF10	1	General purpose register write clock
RF11	3	ALU function select
RF12	3	ALU A, B multiplexer input
RF13	1	Carry input to ALU
RF14	3	I/O Buffer and Memory request and RAM address modifier (0, 32, 64)
RF15	2	Special function decode
RF16	1	Read/Write Select
RF17	1	Spare

Although the core processor architecture is generally conventional, it does contain two somewhat unusual features that are extremely useful in supplementing certain signal processing algorithms: (1) Certain of the ALU output bits can be individually specified as control RAM branch bits. (Usually only sign, carry out, overflow and the all-zeros conditions are used to effect branches.) (2) The microcode can specify that certain of the bits of the data RAM address generated by the processor be complemented, thereby in effect, allowing the microcode to modify the processor-generated base address. The utility of these features will become apparent in the subsequent discussion.

3.1.1 I/O BUFFER AND CONTROL RAM SEQUENCER

As already noted, the core processor can be implemented, with one exception, using chips currently being developed for the FTSC. The FTSC chip designs have all been documented and need not be described in detail here. The one exception, the I/O Buffer and Control RAM Sequencer chip, was developed specifically for the WBSP. This chip is described in the following paragraphs.

3.1.1.1 I/O Pin Designations

Seventy-five pins are needed for the I/O Buffer and Control RAM Sequencer. The assignments for these pins are listed in Table 3-2.

3.1.1.2 General LSI Specifications

The requirements imposed on the I/O Buffer and Control RAM Sequencer are as follows:

1) Facilitate loading the control RAM from the system bus, independent of RALU and scratchpad RAM.
2) Provide buffer registers for the data and address buses.

Table 3-2

I/O BUFFER AND CONTROL RAM SEQUENCER PIN DESIGNATIONS

Quantity	Label	Mnemonic
8	Address Bus	MAB0-7
11	Control Bus	REQ ACK R/W MYADD MXACK MTIYHA1 IMODEN MCLK MODCLR GDEDCA GDEDCD
8	Data Bus (Interface)	MDB0-7
8	Control RAM Enables	CEN0-4 Write en. ADD STB PROCEN
8	Control RAM Address	ROADD0-7
7	Branch Variables	Zero BC0 B0 B01 B5 B6 Interrupt
5	Microcode	RF1, RF2(Bits 6,7)
8	Data Bus (Processor)	PDB0-7
4	RAM Address In	RAADDI0-4
4	RAM Address Out	RAADDO0-4
4	Power	

3) Provide microprogram control logic necessary to operate the processor.
4) Provide the interface between the bus interface chips and the processor itself.
5) Coordinate interprocessor communication.
6) Provide RAM scratchpad address modifications.

3.1.1.3 Functional Block Descriptions

The following is a brief description of each of the functional blocks shown in the I/O Buffer and Control RAM Sequencer block diagram (Figure 3-3):

1) Input Data Buffer	Provides 1 word of data storage (8 bits). This buffer is used for information transfer from the system bus to the processor. It is used to load the control RAM and serves as an input register for the RALU and scratchpad memory. When data are entered into the buffer, a flag is set to interrupt the processor.
2) Output Data Buffer	Provides 1 word of data storage (8 bits). This buffer's purpose is to hold data that are being transferred from the RALU to the system bus.
3) Control Logic	This block controls the distribution of the processor's control signals.
4) Address Buffer	Provides 2 words (16 bits) of address storage. This information is provided by the address interface connected to the system bus. The control logic provides the signals necessary to operate the buffer.
5) Hard/Soft Instruction Decode	This block detects the module's hard and soft addresses. Once it has detected its hard address, it starts the control RAM address and frame counter. Information for this block is provided by the address buffer.

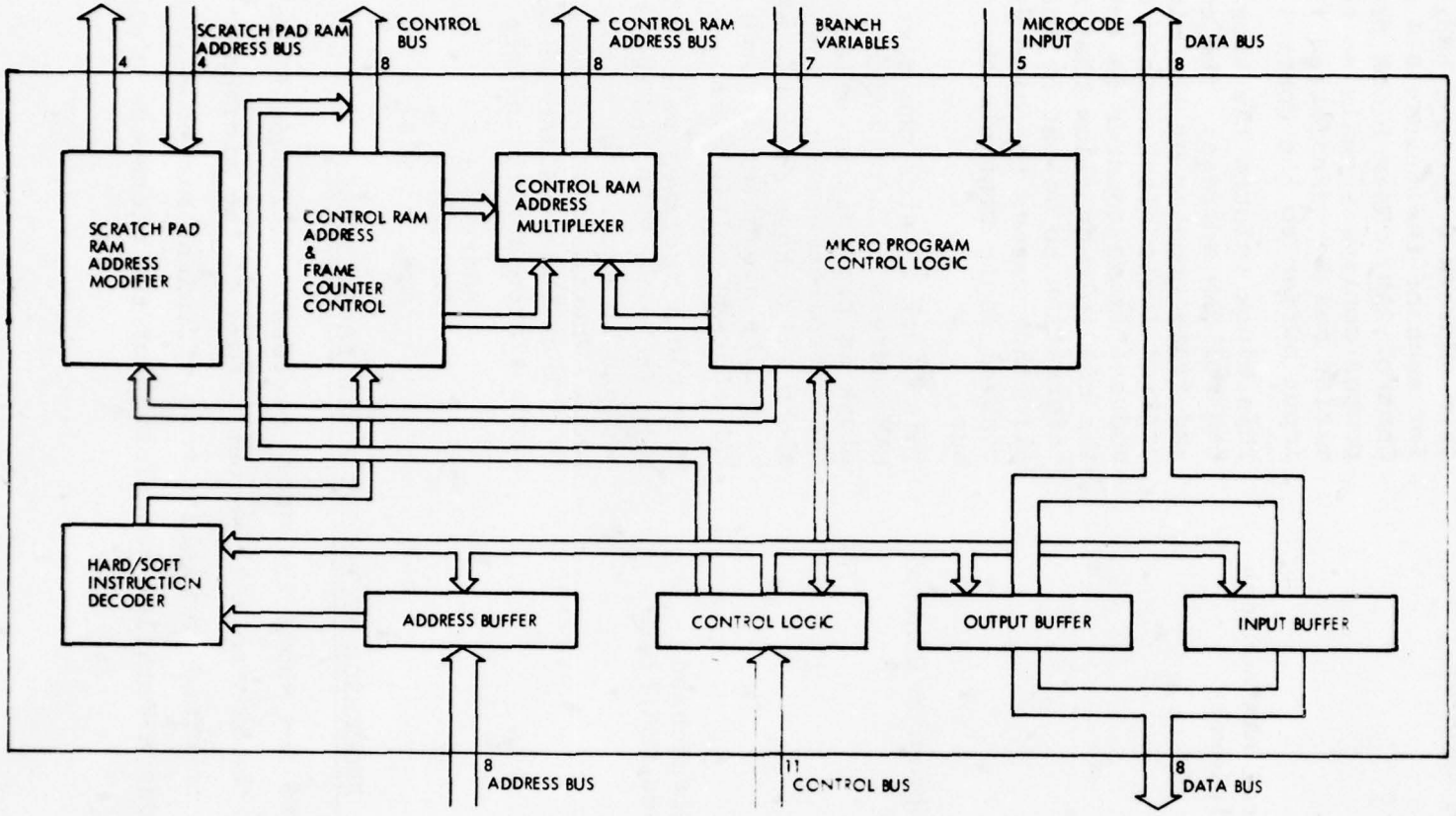


Figure 3-3 I/O BUFFER AND CONTROL RAM SEQUENCER

6) Control RAM Address & Frame Counter	Once activated, this block sequences from address 0 through address 255 for each of the frames 0-4 of the Control RAM. This block provides the proper control signals so that information can be transferred from the input buffer to the control RAM.
7) Control RAM Address Multiplexer	This block selects the source of the control RAM address: the control RAM and frame counter or the microprogram control logic block. Only when the address frame counter is active does the multiplexer allow that address information to be put on the bus. In all other cases, the microprogram control logic controls the address bus.
8) Microprogram Control Logic	This block creates the next control RAM address. Information to this block is provided by selected bits of the microinstruction. These bits along with those from the branch variables are gated through the control RAM address multiplexer.
9) RAM Scratchpad Address Modifier	This block receives selected address bits from the RALU and modifies the address in accordance with the microcode. The modified address is used to select the next RAM scratchpad word. Three conditions are allowed: Address + 0 Address + 32 Address + 64

3.2 CORE PROCESSOR MICROCODE FIELD DEFINITIONS

Figure 3-4 shows the core processor microcode fields as they appear at the control RAM output. Some of the microcode fields have been combined to yield a more efficient microcode. Table 3-3 provides a detailed description of the microcode fields.

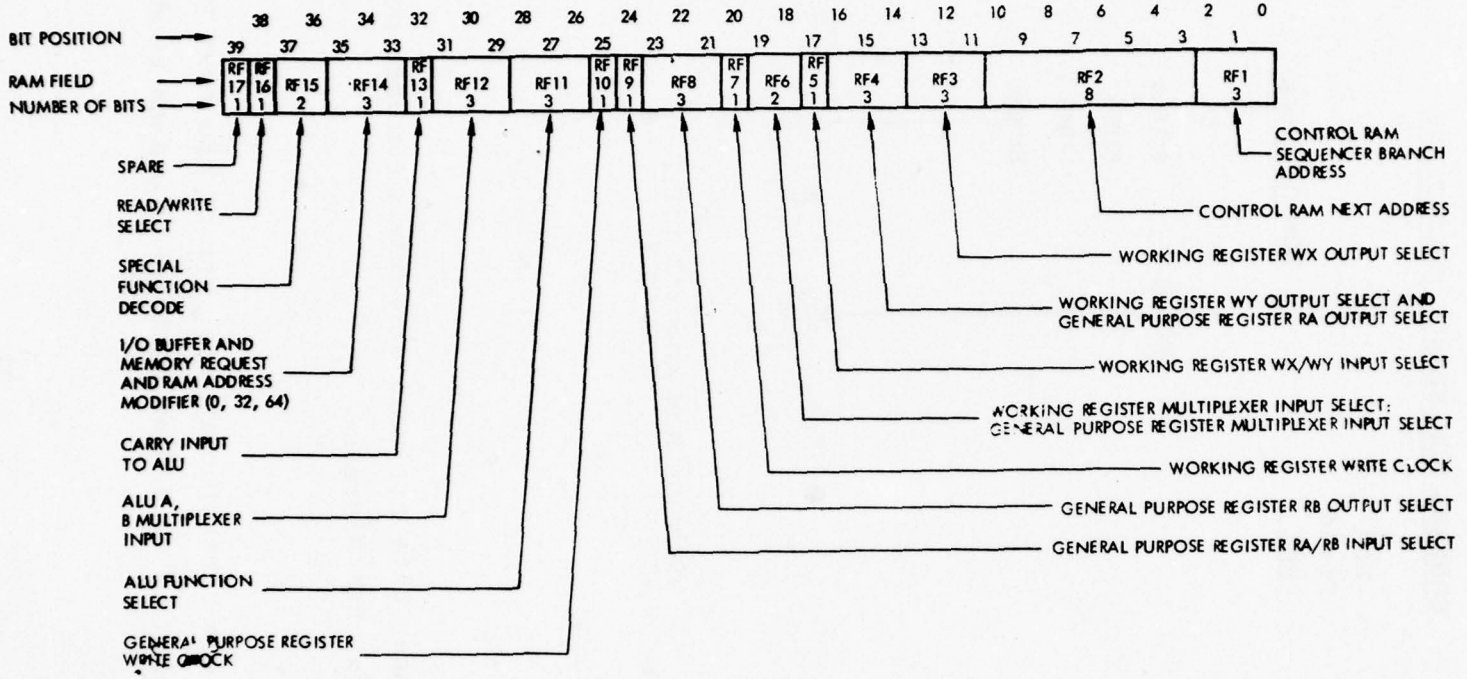


Figure 3-4 BASIC PROCESSOR MICROCODE CONFIGURATION

Table 3-3

MICROCODE FIELD DEFINITIONS

CONTROL ROM BRANCH CONDITIONS

ROM Sequencer Address	ROM Field RF1	ROM Address 5	6	7
0 0 0		ROMA5	ROMA6	ROMA7
0 0 1			SUM6	SUM7
0 1 0			SUM5	SUM6
0 1 1			ROMA6	SUM0
1 0 0				SUM1
1 0 1				CARRY OUT
1 1 0				SUM ZERO
1 1 1				INTERRUPT

CONTROL ROM ADDRESS FIELD

ROM Field- RF2

A0
A1
A2
A3
A4
A5

These bits are used directly to create part of the next ROM Address.

A6
A7

These bits specify the rest of the next ROM address which is dependant on the ROM branch conditions.

Table 3-3 (Cont'd.)

WORKING REGISTER WX OUTPUT SELECT

ROM Field- RF3	Register Select	Assignment
0 0 0	Working Register 0	WR0
0 0 1	Working Register 1	WR1
0 1 0	Working Register 2	WR2
0 1 1	Working Register 3	WR3
1 0 0	Working Register 4	Extension Register
1 0 1	Working Register 5	Memory Data
1 1 0	Working Register 6	Memory Address
1 1 1	Working Register 7	Program Counter

TABLE 3-3 (Cont'd.)

WORKING REGISTER WX/WY INPUT SELECT

ROM Field- RF5	Assignment
0	WX.
1	WY

GENERAL PURPOSE REGISTER RA/RB INPUT SELECT

ROM Field- RF9	Assignment
0	RB
1	RA

Table 3-3 (Cont'd.)

WORKING REGISTER WY OUTPUT SELECT AND GENERAL PURPOSE REGISTER RA OUTPUT SELECT

ROM Field- RF4	Register Select	Assignment
0 0 0	Register Select 0	WY0 + R0
0 0 1	Register Select 1	WY1 + R1
0 1 0	Register Select 2	WY2 + R2
0 1 1	Register Select 3	WY3 + R3
1 0 0	Register Select 4	WY4 + R4
1 0 1	Register Select 5	WY5 + R5
1 1 0	Register Select 6	WY6 + R6
1 1 1	Register Select 7	WY7 + R7

Table 3-3 (Cont'd.)

GENERAL PURPOSE REGISTER ARRAY INPUT MULTIPLEXING

ROM Field- RF6	Register Input	End Bits Input
0 0	Working Register Y	NONE
0 1	SUM Bus	NONE
1 0	SUM Bus Right Shifted by one	Determined by end condition bits
1 1	SUM Bus Left Shifted by one	Determined by end condition bits

WORKING REGISTER ARRAY INPUT MULTIPLEXING

ROM Field- RF6	Register Input	End Bits Input
0 0	SUM Bus	NONE
0 1	Working Register Y	NONE
1 0	Working Register Y Right Shifted by one	Determined by end condition bits
1 1	Working Register Y Left Shifted by one	Determined by end condition bits

Table 3-3 (Cont'd.)

WORKING REGISTER WRITE CLOCK

ROM Field- RF7	Assignment
0	Write Disable
1	Write Enable

GENERAL PURPOSE REGISTER WRITE CLOCK

ROM Field- RF10	Assignment
0	Write Disable
1	Write Enable

Table 3-3 (Cont'd.)

GENERAL PURPOSE REGISTER RB OUTPUT SELECT

ROM Field- RF8	Register Select	Assignment
0 0 0	General Purpose Reg 0	R0
0 0 1	General Purpose Reg 1	R1
0 1 0	General Purpose Reg 2	R2
0 1 1	General Purpose Reg 3	R3
1 0 0	General Purpose Reg 4	R4
1 0 1	General Purpose Reg 5	R5
1 1 0	General Purpose Reg 6	R6
1 1 1	General Purpose Reg 7	R7

Table 3-3 (Cont'd)

ROM Field- 11 ALU FUNCTIONS

SELECT	CARRY IN	
S0 S1 S2	CIN = 0	CIN = 1
0 0 0	AB	AB
0 0 1	A + B	A + B
0 1 0	A'B	A'B
0 1 1	A ⊕ B	A ⊕ B
1 0 0	A plus B	A plus B plus 1
1 0 1	A minus B minus 1	A minus B
1 1 0	B minus A minus 1	B minus A
1 1 1	$\overline{B}$ (1's complement)	-B(2's complement)
B = 0		
0 0 0	0's	0's
0 0 1	A	A
0 1 0	$\overline{A}$ (1's complement)	$\overline{A}$ (1's complement)
0 1 1	A	A
1 0 0	A	A plus 1
1 0 1	A minus 1	A
1 1 0	$\overline{A}$ (1's complement)	-A (2's complement)
1 1 1	1's	0's

Table 3-3 (Cont'd.)

ALU INPUT MULTIPLEXING

ROM Field- RF12			Assignment	
A	B	C	ALUA	ALUB
0	0	0	GPRB	WRX
0	0	1	GPRA	WRX
0	1	0	GPRB	WRXLS
0	1	1	WRX	0's/2
1	0	0	GPRB	GPRA
1	0	1	WRX	GPRA
1	1	0	GPRB	0's/2
1	1	1	GPRA	0's/2

A — ALU B LEG SELECT

B, C — ALU A LEG SELECT

Table 3-3 (Cont'd.)

CARRY INPUT TO ALU

ROM Field-RF13	Assignment
0	No Carry
1	Carry

I/O BUFFER & MEMORY REQUEST

ROM Field-RF14	Assignment
000	Address Modifier Disable (RAM ADDRESS Unmodified)
001	Increment by 32
010	Increment by 64
011	Undefined
100	Input Request
101	Output Request
110	Undefined
111	ALU Function (Default Value)

SPECIAL FUNCTION DECODE

ROM Field-RF15	Assignment
00	No Operation
01	Reset Overflow Flip Flop
10	Set Overflow Flip Flop
11	Enable Carry Flip Flop

READ/WRITE SELECT

ROM Field-RF16	Assignment
0	Write
1	Read

SPARE

ROM Field-RF17	Assignment
0	Undefined
1	Undefined

3.3 CORE PROCESSOR UTILIZATION AS A DECODER

The core processor is used without modification to implement the Encoder/Decoder processor in the WBSP. In order to assess its suitability for this purpose, the entire microcode sequence was defined for three different decoding algorithms: the Viterbi decoding algorithm for a constraint-length 7, rate 1/2 convolutional code; the dual-3 Viterbi decoding algorithm; and a constraint-length 10, rate 1/2 convolutional code feedback decoding algorithm. The constraint-length 7 Viterbi decoding algorithm was selected because it was judged to be the most complex algorithm that might be implemented on board a satellite. It now appears that the dual-3 decoding algorithm and either the feedback decoding algorithm or a Reed-Solomon code decoding algorithm of comparable complexity will actually be implemented. Nevertheless, the following paragraphs demonstrate the versatility of the proposed means of implementing these functions. Since, in particular, the constraint-length 7 Viterbi decoding algorithm can be handled at the rates of concern here using the proposed approach, it can be concluded that any decoding requirement placed on the WBSP can be similarly accommodated.

3.3.1 THE VITERBI DECODER METRIC PROCESSOR

Two decoder processors are used to implement the constraint-length 7 Viterbi decoding algorithm. The metric processor is described first; the path processor is described in paragraph 3.2.2. It is assumed that soft-decision (3-bit quantized) data is deposited in the processor's input buffer. The data representing the first decision associated with a given branch is in bit positions 1-3, and those representing the second decision are in bit positions 5-7 of the 8-bit input word. The result of each metric processor

decision is stored in the output buffer. These decision bits are packed eight to a word to be transferred to the path processor. This output need only be fetched by the bus controller eight times during each information bit period, in order to preserve all sixty-four decisions.

The utilization of the RALU registers is summarized in Table 3-4. (The eight general-purpose registers are denoted R_0-R_7, and the four working registers W_0-W_3; E, A, P, and M denote, respectively, the extension, address, program counter, and memory data registers, and I denotes the interface buffer register.) The storage memory is organized as follows: the first and third 64-word quadrants provide two complete metric memories. One memory is read from and the other written to during one decoding cycle and the two roles reversed during the following cycle. The second quadrants are used to store the two-bit patterns associated with each of the sixty-four branches. Since unused memory is available, this information is duplicated in the fourth quadrant. This duplication allows the appropriate branch information to be fetched from the same relative address regardless of which of the two metric memories is being used.

The microinstructions needed to determine the sixty-four path decisions and to update and normalize the path metrics following each new received branch are itemized in Figure 3-5. The meaning of the abbreviations used there are defined in Table 3-5. A total of 1234 clock cycles (with two cycles allowed for each memory access) are required for each information bit. Present estimates are that the processor can operate using a 150 nsec. clock. If this is the case, a single metric processor could decode information at a rate of roughly 5400 bps.

Table 3-4

VITERBI DECODER REGISTER UTILIZATION

Register	Metric Processor	Path Processor
R_0	PATH DECISIONS	VOTER ON OUTPUT BIT
R_1	10000000	SCRATCH PAD
R_2	00001110	SCRATCH PAD
R_3	00001111	SCRATCH PAD
R_4	NORMALIZATION CONSTANT (INITIALIZED TO 00000000)	DECODED BIT COUNTER
R_5	S_i (I^{TH} METRIC)	DECODED OUTPUT BITS
R_6	$S_i + 32$	11111100
R_7	$S_i - S_{i+32}$	INPUT DATA
W_0	$\overline{\Delta S}_i$	POINTER 1
W_1	$\overline{\Delta S}_{i+32}$	POINTER 2
W_2	$\overline{\Delta S}_i$	POINTER 3
W_3	$\overline{\Delta S}_{i+32}$	OLDEST PATH-SEGMENT COUNTER
E	BRANCH	
M	SCRATCH PAD	

Table 3-4 (Cont'd.)

VITERBI DECODER REGISTER UTILIZATION

Register	Metric Processor	Path Processor
A	SOURCE ADDRESS FOR S_i (INITIALIZED TO 00000000)	BASE ADDRESS (INITIALIZED TO 00000000)
P	DISTINCTION ADDRESS FOR AUGMENTED S_i (INITIALIZED TO 100000000)	COMPETING ADDRESS (INITIALIZED TO 100000000)

Figure 3-5

METRIC PROCESSOR MICROROUTINE

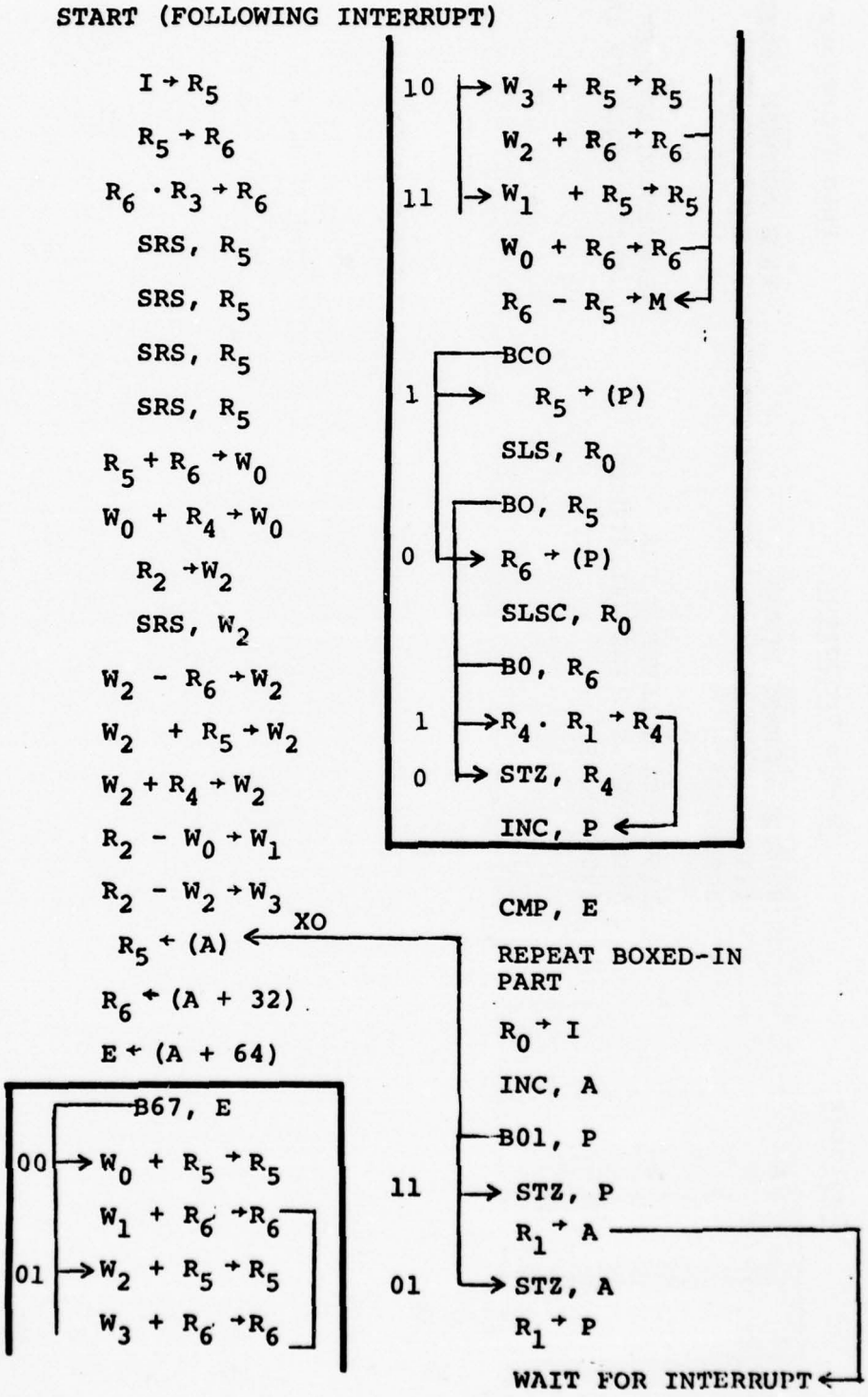


TABLE 3-5

MICROCODE DEFINITIONS

ABBREVIATION	DEFINITION
R → S	Store contents of Register R into Register S.
R → (S)	Store contents of Register R into memory location specified in Register S.
R ← (S)	Load Register R with the contents of the memory location specified in Register S.
R + S → Q	Add (2's comp.) the contents of registers R and S and store the result in Register Q.
R · S → Q	Store in Register Q the logical "and" of the contents of registers R and S.
R ∨ S → Q	Store in Register Q the logical inclusive "or" of contents of registers R and S.
R ⊕ S → Q	Store in Register Q the logical exclusive "or" of contents of registers R and S.
R - S → Q	Subtract (2's complement) contents of Register S from Register R and store result in Register Q.
BCO	Branch on the ALU's carry-out bit.
B_{ij}, R	Branch on bits i, j in Register R.
B_j, R	Branch on bit j in Register R.
CMP, R	Replace contents of Register R by its one's complement.
DEC, R	Decrement (by one) the contents of Register R.
INC, R	Increment (by one) the contents of Register R.
JNZ, R	Jump unless Register R contains only zeros.
JZE, R	Jump if Register R contains only zero.
SLR, R	Cyclically rotate contents of Register R one position to the left.
SLS, R	Shift contents of Register R one position left; shift zero into least significant (right-most) bit position.
SRR, R	Cyclically rotate contents of Register R one position to the right.
SRS, R	Shift contents of Register R one position to the right; shift a zero into most significant (left-most) bit position.
SLSC, R	Shift contents of Register R one position to the left and shift a one into the least significant bit position.
STZ, R	Store all zeros into Register R.

3.3.2 VITERBI DECODER PATH PROCESSOR

The path processor random-access memory is functionally divided into sixty-four segments. Each segment, consisting of four consecutive addresses, is used to store the thirty-two bits representing one of the sixty-four contending paths. Double storage of this information is obviated by dynamically reassigning path addresses. Initially, for example, path 0 is stored in locations 0, 1, 2 and 3 and path 32 is stored in locations 128, 129, 130 and 131. The first decision made by the metric processor determines whether the new path 0 is to be defined by the old path 0 (augmented by 0) or by the old path 32 (augmented by 1). Similarly, the second decision determines whether the new path 1 is to be defined by the old path 0 or by old path 32. Once the results of the first two decisions are available, the old paths 0 and 32 are no longer needed. Thus, new path 1 can be stored in locations 128, 129, 130 and 131.

Since the path addresses are continually changing, it is obviously necessary to keep a record of the current assignments. This is done quite simply by defining two pointers (pointers 2 and 3 in Table 3-4), one of which is used to increment the address to proceed from one path to the next and the second to complement one of the bits in the address in order to generate the address of the competing path. These pointers are then shifted one position to the right (modulo 6) following each pass through the entire path memory. Initially, then, pointer 2 is set to 100000 and pointer 3 to 000001. (The two least significant bits are irrelevant to this discussion and are hence ignored.) The base address A (initially 000000) is added to pointer 2 in order to define the address of the competing path. (Addition here is defined as addition, modulo 63, with the result interpreted as an

integer in the interval (1, 63).) The next base address (designating the location of one of the two paths involved in the next two decisions) is generated by adding (modulo 63) pointer 3 to the current base address, etc. (Thus, initially, the base addresses are generated in the sequence 0, 1, 2, ..., and the competing addresses in the sequence 32, 33, 34, ...). After each path through the entire memory (producing one decoded bit), both pointers are rotated one position to the right, modulo 6. Thus, pointer 2 is set to 010000 and pointer 3 to 100000 during the second pass (producing the base address sequence 0, 32, 1, 33, 2, 34, ..., and the competing address sequence 16, 48, 17, 49, 18, 50, ...). It is not difficult to verify that this address-generating algorithm does indeed produce the proper base sequence and the relationship between the base competing addresses. Since no location is stored into until the previous contents of that location are no longer of interest, double storage is not necessary. Moreover, the two address sequences are easily generated using this simple algorithm.

A third pointer (pointer 1 in Table 3-4) and a modulo-4 counter are used to keep track of the location of the oldest bit in each path. The pointer is rotated one position to the right (modulo 8) following each pass through the memory; the counter is augmented by one after each eight pointer rotations. The counter thus indicates which of the four memory locations corresponding to any given path contains the oldest bit and the pointer identifies the position of the oldest bit in that location. Allowing the location of the oldest bit to change in this manner eliminates the need to shift each 32-bit path during each pass through the memory. Since it is obviously much easier to rotate an eight-bit pointer and to increment a modulo-four counter than to shift 64

32-bit paths, this procedure significantly reduces the amount of time needed to decode each bit.

The microinstructions needed to manipulate the path memory and to determine the decoded output (using a majority vote on the oldest bit in each path) are listed in Figure 3-6. (The abbreviations used there are those defined in Table 3-5.) The sixty-four metric processor decisions corresponding to one information bit are presented to the path processor in eight eight-bit words (cf, Section 2). The decoded output bits are packed eight to a word. If two 150 nanosecond clock cycles are required for each memory access and one clock cycle for all other microinstructions, a maximum of 1717 clock cycles are needed to decode an information bit. Thus, the path processor can operate at information rates of up to 3800 bps.

3.3.3 DUAL-3 VITERBI DECODER IMPLEMENTATION

The dual-3 Viterbi decoding algorithm, as implemented by a decoder processor, can be conveniently divided into three phases. During phase 1, the eight 3-bit soft-decision characters representing the first symbol of the branch being processed are received and stored in registers R_0 through R_7. (For purposes of this discussion, it is assumed that these characters are transferred as 8 data words.) Each of the 8 path metrics is then added to each of these 8 "delta-metrics" to form 64 partially augmented metrics, corresponding to the 64 extended paths, which are stored in the data RAM for subsequent processing.

The second phase (phase 2A) starts when the eight 3-bit characters representing the second branch symbol are received and again stored in registers R_0 through R_7. These second delta-metrics are similarly added to the 64 partially augmented metrics

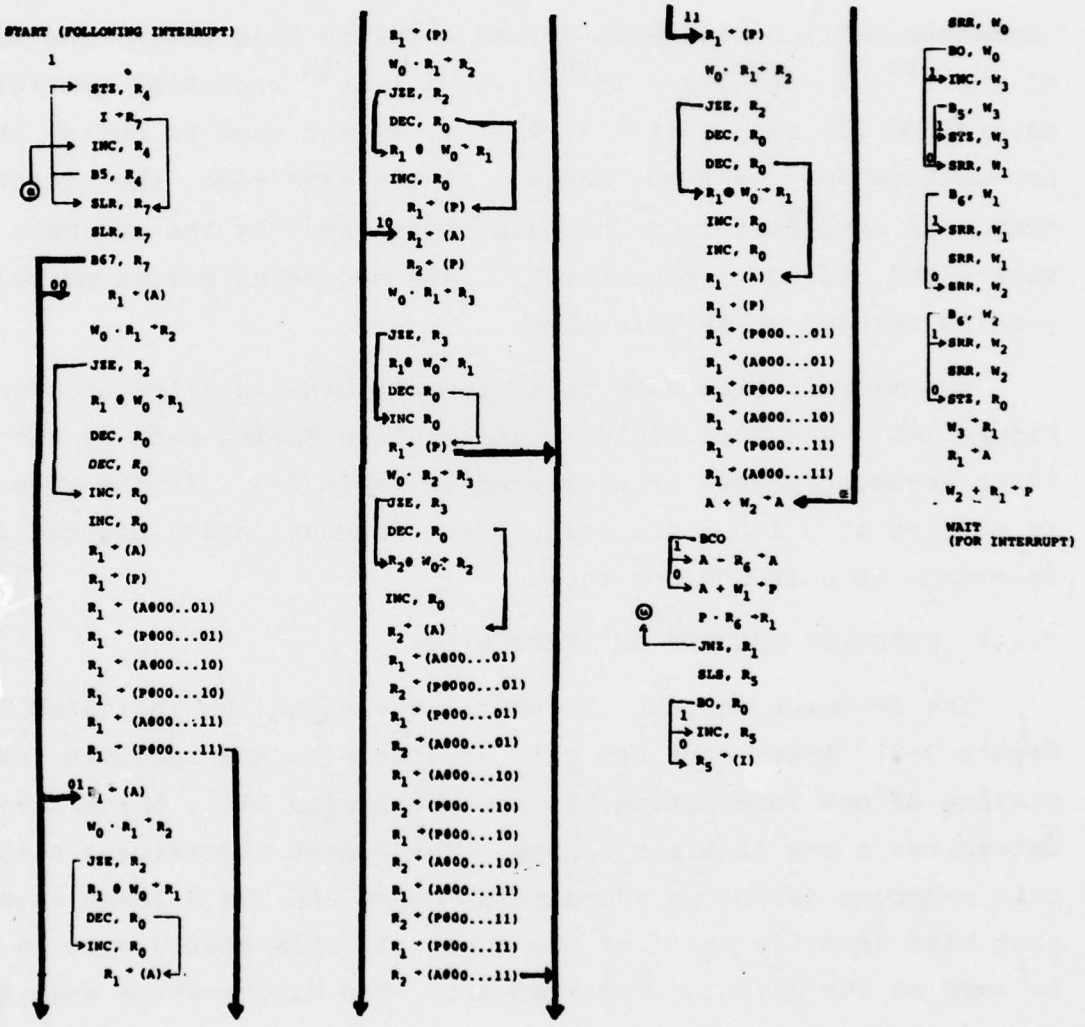


Figure 3-6 PATH PROCESSOR MICROROUTINE

already stored to form the 64 fully augmented metrics corresponding to the 64 contending paths. The order in which this second set of delta-metrics are used is determined by the contents of the "address modifier" section of the data memory. (The data memory partitioning is shown in Table 3-6.)

The third decoding phase (phase 2B) begins as soon as all 64 augmented metrics have been formed. During this phase, the smallest of the i^{th}, $i + 8^{th}$, $i + 16^{th}$, ..., $i + 64^{th}$ augmented metrics is determined for each i (i = 1, 2, ..., 8) and used to define the 8 new metrics and the 8 new paths. At the same time, the oldest symbol is extracted from the path represented by the minimum metric and stored in register R_7. Any necessary metric normalization is carried on at this time.

The microroutine used to implement this algorithm is shown in Figure 3-7. The RALU register utilization during each of the three decoding phases is presented in Table 3-7. If the processor is clocked at a 150 nsec. rate, it can decode dual-3 encoded information at a 10,500 bps rate.

3.3.4 FEEDBACK DECODER IMPLEMENTATION

The feedback decoder implements the algorithm indicated in Figure 3-8. After each new pair of code bits are received (consisting of one information bit and one parity bit), the processor determines a new 11-bit syndrome. The 8 most significant bits of this syndrome define an address in memory and the 3 least significant bits identify which of the 8 bits in this stored word is to be used as the output correction bit. The microroutine used to implement this algorithm is shown in Figure 3-9. The RALU register utilization is described in Table 3-8. (The information and syndrome bit numbers shown there correspond to the numbers indicated in Figure 3-8.) The processor can implement this algorithm at information rates of up to 211,000 bits/second.

Table 3-6
DUAL-3 DECODER MEMORY PARTITIONING

0 0 0 0 0 0 0 0	Path 0 (symbols 1 and 2)
	Path 1 (symbols 1 and 2)
	⋮
Path Memory A	Path 7 (symbols 1 and 2)
	Path 0 (symbols 3 and 4)
	Path 1 (symbols 3 and 4)
	⋮
0 0 1 1 1 1 1 1	Path 7 (symbols 15 and 16)
0 1 0 0 0 0 0 0	Metric 0
Metric Memory	⋮
0 1 0 0 0 1 1 1	Metric 7
0 1 0 0 1 0 0 0	0 0 0 0 0 0 0 0
	0 0 0 0 0 1 0 1
Phase 2A	0 0 0 0 0 0 0 1
Address Modifiers	0 0 0 0 0 1 0 0
	0 0 0 0 0 0 1 0
0 1 0 0 1 1 1 1	0 0 0 0 0 1 1 1
	0 0 0 0 0 0 1 1
	0 0 0 0 0 1 1 0
0 1 0 1 0 0 0 0	Normalizer
Phase 2B	Path Memory A/B Pointer
Variables	Symbol Mask
0 1 0 1 0 0 1 1	1 1 0 0 0 0 0 0
0 1 0 1 0 1 0 0	Unused
0 1 0 1 1 1 1 1	
1 0 0 0 0 0 0 0	Same organization as Path Memory A
Path Memory B	
1 1 0 0 0 0 0 0	$M_0 + \Delta M_0$
	$M_0 + \Delta M_1$
Contending Metrics	⋮
	$M_0 + \Delta M_7$
1 1 1 1 1 1 1 1	$M_1 + \Delta M_0$
	⋮
	$M_7 + \Delta M_7$

TABLE 3-7

DUAL-3 DECODER REGISTER USAGE

REGISTER	PHASE 1	PHASE 2A	PHASE 2B
R_0	ΔM_{00}	ΔM_{01}	Scratch Pad
R_1	ΔM_{10}	ΔM_{51}	Index of current best metric
R_2	ΔM_{20}	ΔM_{11}	Minimum metric found in this sequence (initialized to 11111111)
R_3	ΔM_{30}	ΔM_{41}	Metric pointer (initialized to W_3)
R_4	ΔM_{40}	ΔM_{21}	Normalizer
R_5	ΔM_{50}	ΔM_{71}	Path Memory A/B Pointer (alternates between 00000000 and 10000000)
R_6	ΔM_{60}	ΔM_{31}	Symbol Mask (alternates between 00000111 and 00111000)
R_7	ΔM_{70}	ΔM_{61}	DECODED SYMBOL
W_0	01000000	01000000	01000000
W_1	Base address for contending metrics (initialized to 11000000)	Base address for contending metrics (initialized to 11000000)	Current metric pointer (used instead of R_3 on odd cycles)
W_2	Index for contending metrics (initialized to 00000000)	Index for contending metrics (loaded from memory)	Index used during metric comparisons
W_3	00001000	00001000	00001000
A	Address Register used to load present metrics	Address Register (used to load W_2)	Address Register (used to load contending metrics)
P	Address Register (used update metrics)	Address Register (used to update metrics)	Address Register (used to update path and metric memories)
M	Scratch Pad	Scratch Pad	Scratch Pad
E	Scratch Pad	Not Used	Scratch Pad

TABLE 3-8

FEEDBACK DECODER REGISTER USAGE

R_0 = Information bits being corrected (4 to -3)

R_1 = Most recent information bits (15 to 8)

R_2 = Next most recent information bits (7 to 0)

R_3 = Most recent (modified) syndrome bits (15 to 8)

R_4 = Next most recent (modified) syndrome bits (7 to 0)

R_5 = Third most recent (modified) syndrome bits (-1 to -8)

R_6 = Correction bits: [(R_5) determines which of the 8 bits is to be used]

R_7 = Counter

A = Memory address register [Loads R_6 from (A)]

M = 00001000

Syndrome memory organization:

8 most recent (modified) syndrome bits → address

3 oldest (modified) syndrome bits → bit

to be stored in given position at that address:

Oldest	Next Oldest	Third Oldest	Use Bit
0	0	0	7
0	0	1	6
0	1	0	5
0	1	1	4
1	0	0	3
1	0	1	2
1	1	0	1
1	1	1	0

Phase 1

Wait for Interrupt	$W_1 \oplus W_2 \rightarrow P$	$M + R_5 \rightarrow E$
$I_i \rightarrow R_0$	$(A) \rightarrow M$	$E \rightarrow (P)$
Wait for Interrupt	$M + R_0 \rightarrow E$	INC W_1
$I_i \rightarrow R_1$	$E \rightarrow (P)$	$W_1 + W_2 \rightarrow P$
Wait for Interrupt	INC W_1	$M + R_6 \rightarrow E$
$I_i \rightarrow R_2$	$W_2 \oplus W_2 \rightarrow P$	$E \rightarrow (P)$
Wait for Interrupt	$M + R_1 \rightarrow E$	INC W_1
$I_i \rightarrow R_3$	$E \rightarrow (P)$	$W_1 + W_2 \rightarrow P$
Wait for Interrupt	INC W_1	$M + R_7 \rightarrow E$
$I_i \rightarrow R_4$	$W_1 \oplus W_2 \rightarrow P$	$E \rightarrow (P)$
Wait for Interrupt	$M + R_2 \rightarrow E$	COMP W_1
$I_i \rightarrow R_5$	$E \rightarrow (P)$	JZE W_1
Wait for Interrupt	INC W_1	COMP W_1
$I_i \rightarrow R_6$	$W_1 \oplus W_2 \rightarrow P$	INC W_1
Wait for Interrupt	$M + R_3 \rightarrow E$	INC W_2
$R_7 \rightarrow I_0$	$E \rightarrow (P)$	INC A
$I_i \rightarrow R_7$	INC W_1	STZ W_1
STZ W_1	$W_1 \oplus W_2 \rightarrow P$	SRSC W_1
SRSC W_1	$M + R_4 \rightarrow E$	SRSC W_1
SRSC W_1	$E \rightarrow (P)$	INC A
STZ W_2	INC W_1	Wait for Interrupt to Start Phase 2A
$W_0 \rightarrow A$	$W_1 \oplus W_2 \rightarrow P$	

Figure 3-7 DUAL-3 DECODER MICROROUTINE

Phase 2A

Wait for Interrupt

$I_i \rightarrow R_0$

Wait for Interrupt

$I_i \rightarrow R_5$

Wait for Interrupt

$I_i \rightarrow R_1$

Wait for Interrupt

$I_i \rightarrow R_4$

Wait for Interrupt

$I_i \rightarrow R_2$

Wait for Interrupt

$I_i \rightarrow R_7$

Wait for Interrupt

$I_i \rightarrow R_3$

Wait for Interrupt

$I_i \rightarrow R_6$

$(A) \rightarrow W_2$

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_0 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_1 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_2 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_3 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_4 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_5 \rightarrow M$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_6$

$M \rightarrow (P)$

INC W_1

$W_1 \oplus W_2 \rightarrow P$

$(P) \rightarrow M$

$M + R_7 \rightarrow M$

$M \rightarrow (P)$

COMP W_1

JZE W_1

COMP W_1

INC W_1

INC A

$W_0 \rightarrow W_1$

STZ R_0

STO, R_2

$W_3 \rightarrow R_3$

$(A) \rightarrow R_4$

INC A

$(A) \rightarrow R_5$

INC A

$(A) \rightarrow R_6$

INC A

$(A) \rightarrow A$

(To Phase 2B)

Figure 3-7 (Cont'd.)

Phase 2B

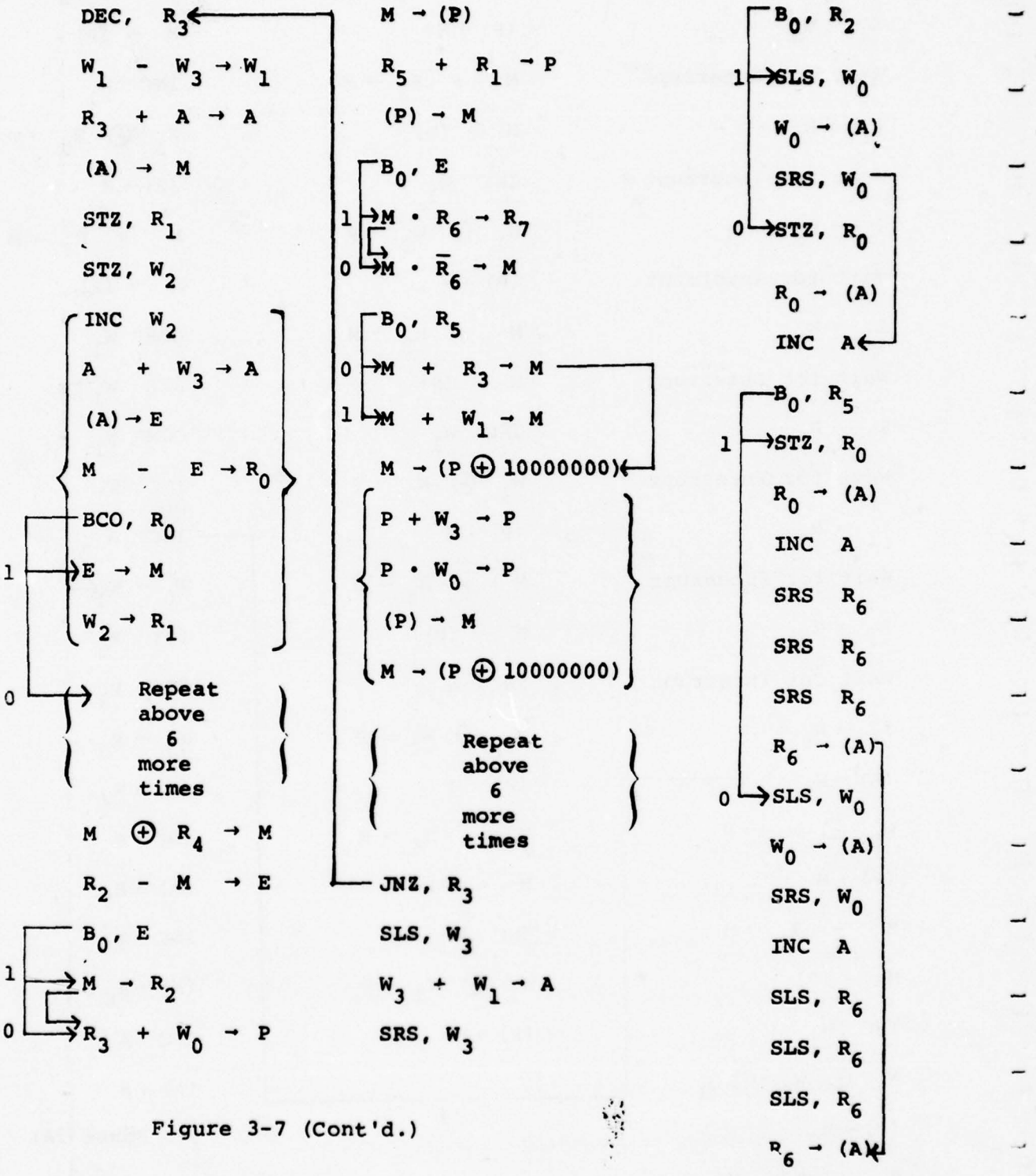


Figure 3-7 (Cont'd.)

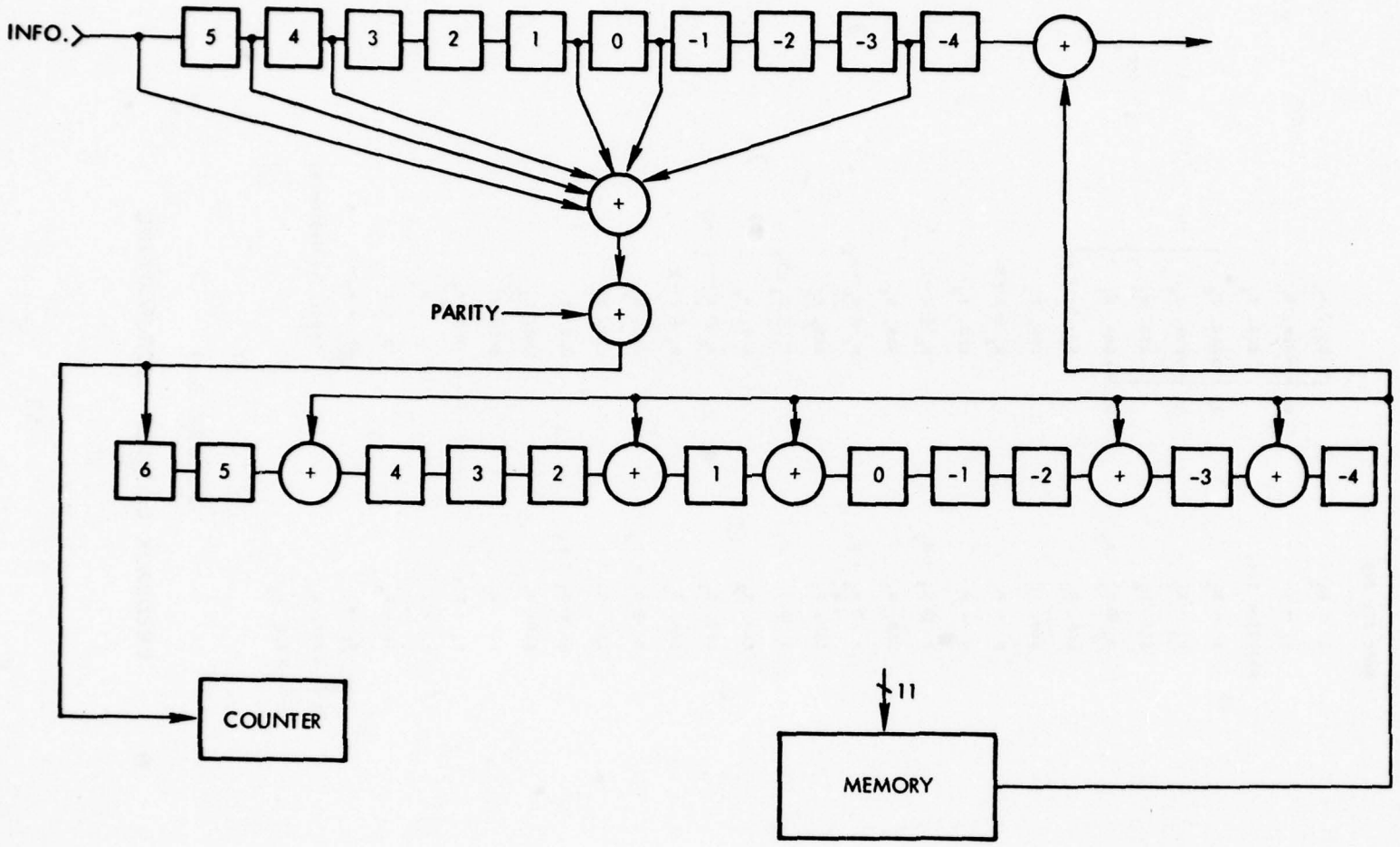


Figure 3-8 FEEDBACK DECODER ALGORITHM

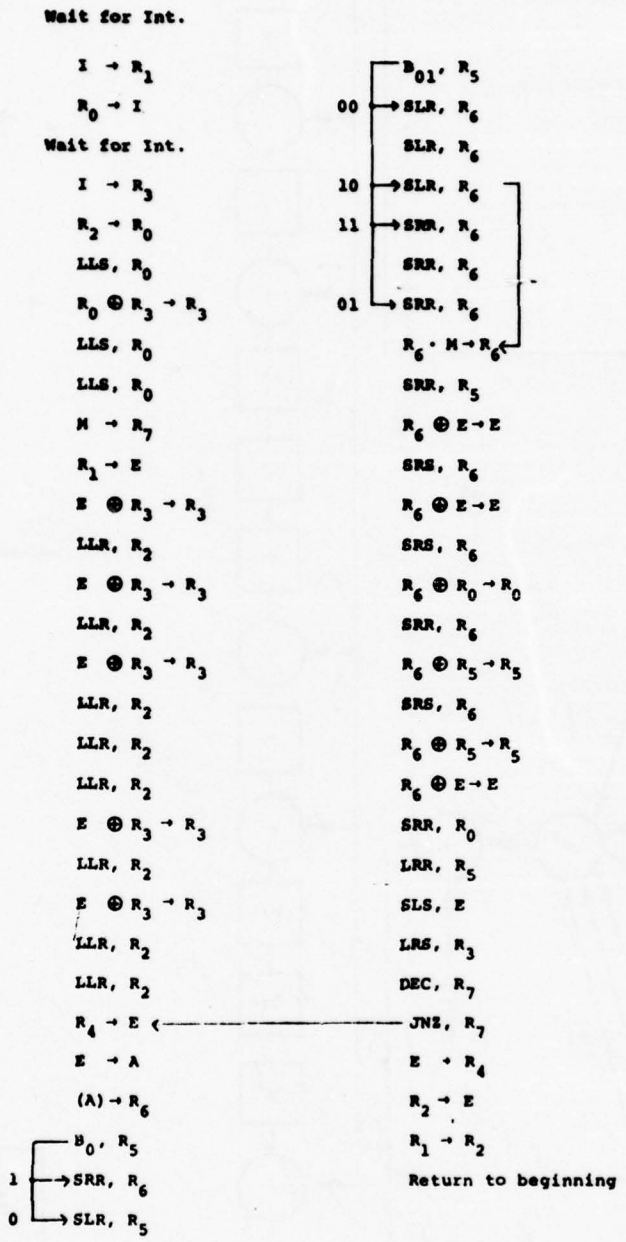


FIGURE 3-9

FEEDBACK DECODER MICROROUTINE

3.3.5 DECODER PROCESSOR SUMMARY

The rates at which the decoder processors can implement each of the algorithms described in the preceeding paragraphs are summarized in Table 3-9. It can be concluded from these investigations that the code processor is capable of implementing any decoding algorithm likely to be required for the SSS satellite at rates well in excess of those rates actually anticipated.

Table 3-9

DECODER PROCESSING RATES

(Information bits/second)

Application	Rate
Metric Processor (K = 7, rate = 1/2)	5,400
Path Processor (K = 7, rate = 1/2)	3,800
Dual-3 Decoder	10,500
Feedback Decoder	211,000

4.0 INTERLEAVER/DE-INTERLEAVER PROCESSOR

4.1 GENERAL DESCRIPTION

The Interleaver/De-interleaver processor consists of a core processor (cf, para. 3.1) augmented with an interleaving memory consisting of 3096 25-bit words (24 information bits plus 1 parity bit). The major variable in this processor design was in the memory size. The core processor can easily handle the computational loads anticipated for it. Any appreciably simplified version of this processor (e.g., making it four bits rather than eight bits wide) would severly reduce its computational margin, however, without significantly improving its reliability. The memory size specified here was arrived at based on reliability and functional partitioning considerations (cf, Section 6).

Since the long-term failure probability of the interleaving memory is not small compared to that of the core processor itself, the Interleaver/De-interleaver processor reliability can be significantly improved by making this memory fault-tolerant. This can be done by adding three spare bits to each word (making the memory 28 bits wide) and using the bit rippler being developed for the FTSC to replace failed bits with these spares. This small added redundancy effectively reduces the hazard rate of the memory by a factor of 36; i.e., from that associated with the 72 1024 x 1-bit chips needed to implement a non-redundant memory to that due solely to the two identical devices needed to implement the rippler switch. This is because the probability of more than three bit failures in a 7-year mission is negligible compared to the probability of a failure in the rippler itself.

Since the information is stored in memory in 25-bit words and since there are 3096 such words, logic is needed to buffer the information in 8-bit bytes and to extend the basic processor's

8-bit address field. In addition, the Interleaver/De-interleaver processor is designed with a single-buffer memory. Thus, memory fetch and store addresses must be constantly shuffled so that no attempts are ever made to write information into a memory location before the information previously stored in that location has been read. The logic needed to accomplish this buffering and address manipulation can all be integrated onto a single CMOS/SOS LSI device. This device, the only special development needed specifically for the Interleaver/De-interleaver processor, is discussed in detail in para. 4.2.

It is, of course, possible to avoid the need for the address shuffling operation mentioned in the previous paragraph by doubling the size of the memory; half the memory could then be used for storing new information and the second half for fetching previously stored information, with the two halves changing roles each frame. The bit rippling scheme discussed earlier could be used to keep the reliability penalty of the doubled memory nearly as small as that of the single memory. The major penalty of doubling the memory, therefore, is in the increased weight and volume of the resulting WBSP. Since the Space Shuttle is to be used to launch the SSS, this penalty may not be that significant. If a special LSI device is developed to handle the other memory control and data buffering functions mentioned above, however, the logic needed to accommodate address shuffling can easily be integrated onto the same device. While such a development is not manditory (a minimal, double-buffered memory control function could be implemented using MSI and SSI logic without unacceptably large reliability penalties), it is particularly appealing when it is recognized that this same device could be used as a memory control chip in the Demodulator processor as well (cf, Section 5). For purposes of this discussion,

therefore, it is assumed that a special memory control chip development will be undertaken and hence that a double-buffered memory is unnecessary.

4.2 INTERLEAVER MEMORY CONTROL CHIP

A functional block diagram of the interleaver memory control chip is shown in Figure 4-1. The data buffer and code checker consists of a 25-bit register used to buffer, code and check data being transferred between the processor and memory. A complete cycle proceeds as follows: Data is fetched from memory; it arrives as a single 25-bit word consisting of 24 information bits and one parity bit. Parity is checked, and if found to be correct, the word is transferred to the processor as three 8-bit bytes in three successive clock pulses. (If a parity violation is discovered, the rippler control logic is activated causing the rippler to be "exercised" and hence to isolate any faulty bit line and ripple in a spare. This latter procedure is identical to that used in the FTSC and is described in detail elsewhere.) Three data bytes are then accepted from the processor, formatted into a 24-bit word to which a parity bit is added, and stored to the memory's next available store address.

The control logic section shown in Figure 4-1 is a relatively straightforward combinational logic circuit used to translate the various input control signals into internal control signals. It also generates the output strobes used to select the appropriate one of the three memory subblocks. The store-address and fetch-address generators and their time-shared stop-address comparator are described in paragraph 4.2.2. The following paragraphs first describe the principle under which they operate.

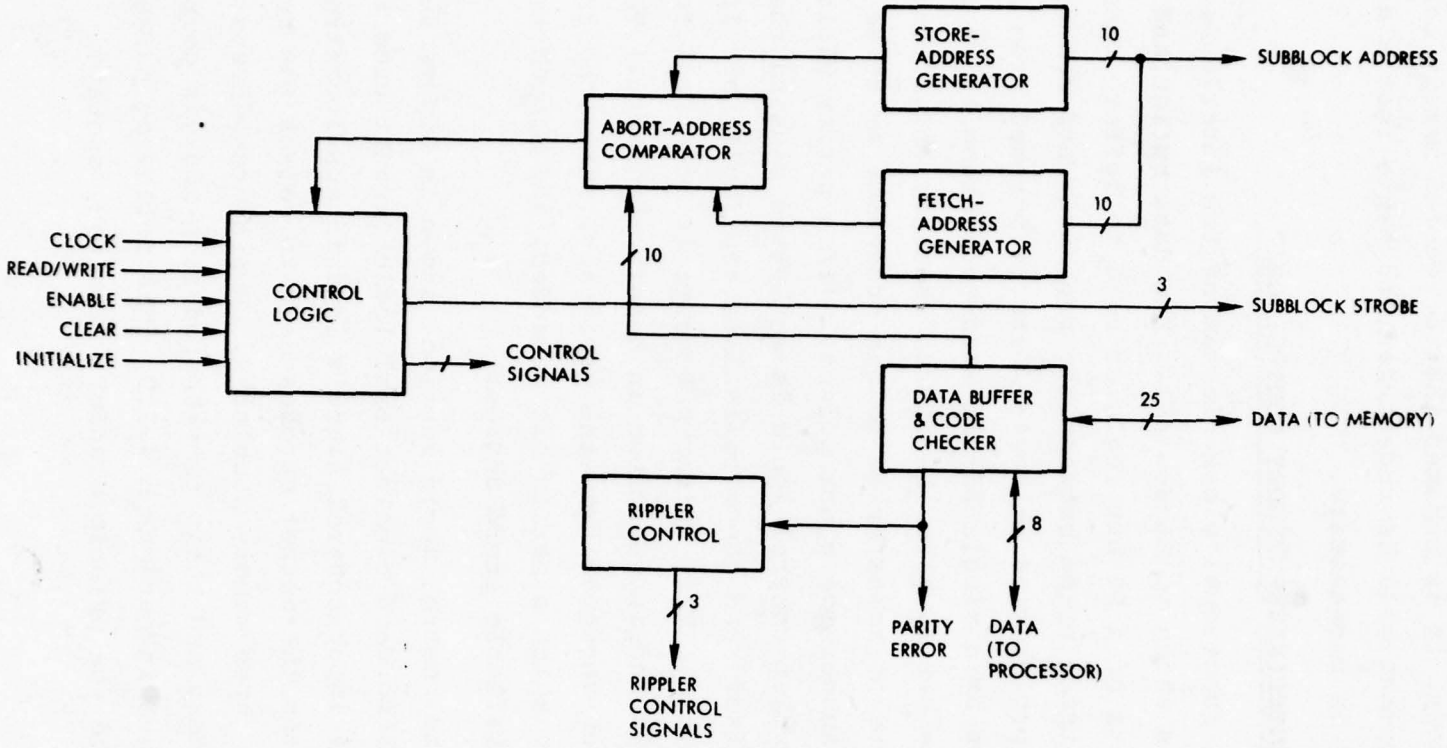


Figure 4-1 INTERLEAVER MEMORY CONTROL

4.2.1 ADDRESS SHUFFLING USING PRIMITIVE ROOTS OVER GF(2^n)

Let α be a fixed element in GF(2^n) and consider the mapping $\beta \rightarrow \beta\alpha$ with β any element in the same field. If α and β are represented as binary n-tuples, then this mapping converts the binary sequence 1, 2, 3, ..., 2^n-1 into the binary sequence α, 2α, 3α, ..., $(2^n-1)\alpha$ (with 1 = 000··01, 2 = 000···010, etc.). Since $\alpha \neq 0$ and since multiplication here is over GF(2^n) all of the n-tuples in this second sequence are unique. Thus, the mapping $\beta \rightarrow \beta\alpha$ interleaves the address sequence 1, 2, 3, ..., 2^n-1. (This is the normal interleaving procedure implemented with pseudo-noise-sequence address generators.)

Now consider the operation of a de-interleaver*. Let the information W_1, W_2,..., W_N be interleaved by the inverse mapping $\beta \rightarrow \alpha^{-1}\beta$ so that it is actually received in the order $W_{\alpha^{-1}1}$, $W_{\alpha^{-1}2}$, ..., $W_{\alpha^{-1}N}$. (For the moment, $N = 2^n-1$.) If this information is stored in memory in "natural" order (i.e., if $W_{\alpha^{-1}i}$ is stored in location i), it can be successfully de-interleaved by fetching it in the order $\alpha 1$, $\alpha 2$, ..., αN. This follows because $W_{\alpha^{-1}i}$ is stored in location i; thus W_i must be stored in location αi. If a new block of information $W'_{\alpha^{-1}1}$, $W'_{\alpha^{-1}2}$, ..., $W'_{\alpha^{-1}N}$ is being received and must be stored at the same time the previous block is being fetched and if only N memory locations are to be used, the new information $W'_{\alpha^{-1}1}$ must be stored in location $\alpha 1$ since that is the only location initially available (after W_1 has been fetched). By extension, $W'_{\alpha^{-1}i}$ must be stored in location αi. Thus, this second block of information is de-interleaved by accessing memory in the order $\alpha^2 1$, $\alpha^2 2$, ..., $\alpha^2 N$. That is, since $W'_{\alpha^{-1}i}$ is now

*Although the discussion here concentrates on the de-interleaver, it is apparent that the same principle applies to the interleaver as well with α replaced everywhere by α^{-1}.

stored in location αi, W_i' must be in location $\alpha^2 i$. It thus follows that if new information is to be stored in those locations just vacated as each word of the previous block is fetched, then the ℓ^{th} block should be accessed in the order $\alpha^\ell 1$, $\alpha^\ell 2$, ..., $\alpha^\ell N$.

If N is less than 2^n-1, the above procedure need only be altered as follows: During the ℓ^{th} frame, abort all stores to addresses a for which $\alpha^{-\ell} a > N$ and abort all fetches from addresses b for which $\alpha^{-(\ell-1)} b > N$. Recall that in the ℓ^{th} frame, the addresses are read in the order $\alpha^\ell \cdot 1$, $\alpha^\ell \cdot 2$, ... Since $\alpha^{-\ell} a > N$ if and only if $a = \alpha^\ell \cdot i$ with $i > N$, this rule prevents stores only to the last 2^n-1-N addresses that would normally have been accessed. Consequently, whenever an access is aborted, the next address in the sequence can immediately be accessed, regardless of whether the aborted address is a fetch or a store address; there is no danger that an attempt will be made to store into a location before its previous contents have been fetched.

The address generator described in the following paragraphs is based in the principles just outlined. In this implementation, α is chosen to be a primitive element in $GF(2^n)$ (although the concept is equally valid for non-primitive elements) and shift-registers are used to generate successive powers of α. Use is made of the fact that, for any ℓ, α^ℓ can be expressed in the form $\alpha^\ell = a_0 + a_1\alpha + a_2\alpha^2 + \ldots + a_{n-1}\alpha^{n-1}$ with a_i in GF(2).

4.2.2 ADDRESS GENERATOR IMPLEMENTATION

An address generator block diagram is shown in Figure 4-2. For exposition purposes, the address generator depicted is for n=3 ($N \leq 7$); the generalization to the case of interest here (n=10) is obvious.

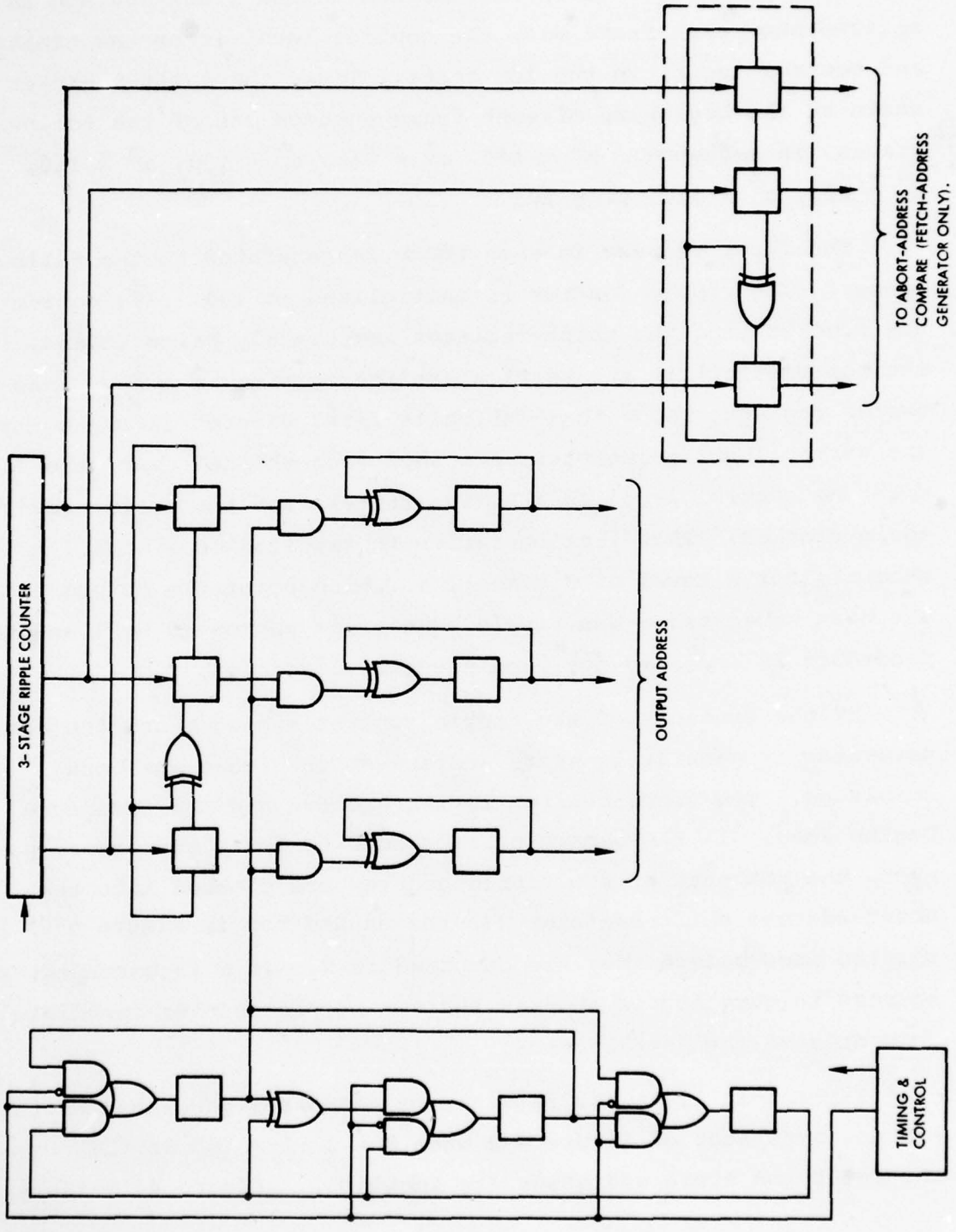


Figure 4-2 ADDRESS GENERATOR

The vertical shift-register shown at the left in Figure 4-2 is a frame counter. It is initialized to the state 100 and is shifted once each frame with the control level (from the timing and control logic) in the low state. Thus, the shift-register state at the beginning of each frame assumes one of the following states, in sequence: $\alpha^0 = 100$, $\alpha^1 = 010$, $\alpha^2 = 001$, $\alpha^3 = 110$, $\alpha^4 = 011$, $\alpha^5 = 111$, $\alpha^6 = 101$.

The first address in each frame is generated in the following manner: The ripple counter is initialized to 100. Its contents are clocked into the shift-register immediately below it and, if the top-most bit in the vertical shift-register is a one, into the buffer register below that (which is first cleared). The horizontal and vertical shift-registers are then each shifted once (the latter with the control level in the high state), and the buffer register again clocked. This last operation is repeated once more (in general, for a total of n times), at which point the output address has been generated. The ripple counter is augmented by 1 and the procedure is repeated for each successive address.

If the contents of the ripple counter exceed N and the address generator is generating store addresses, the frame has been completed. The frame counter is incremented and the procedure begins anew. If the generator is producing fetch addresses, however, the contents of the ripple counter are clocked into the abort-address shift-register (in the dashed box in Figure 4-2) and shifted once before they are compared to N. If N is exceeded, the address in question is aborted and the ripple counter immediately augmented to the next address.

Table 4-1a lists the sequence of addresses generated by the address generator of Figure 4-2 when $N = 2^n - 1 = 7$. In this case, the fetch and store addresses are identical. Figure 4-1b lists the

TABLE 4-1

ADDRESS GENERATOR ADDRESS SEQUENCES

a) $N = 2^n - 1 = 7$

ADDRESS NUMBER	FRAME NUMBER						
	1	2	3	4	5	6	7
1	1	2	4	2	6	7	5
2	2	4	3	6	7	5	1
3	3	6	7	5	1	2	4
4	4	3	6	7	5	1	2
5	4	1	]	3	2	6	7
6	6	7	5	1	2	4	3
7	7	5	1	2	4	3	6

b) N = 5

ADDRESS NUMBER*	FRAME NUMBER													
	1		2		3		4		5		6		7	
	F	S	F	S	F	S	F	S	F	S	F	S	F	S
1	1	1	2	2	4	4	3	3	6	6	7	7	5	5
2	2	2	4	4	3	3	6	6	7	7	5	5	1	1
3	4	3	3	6	6	7	7	5	5	1	1	2	2	4
4	5	4	1	3	2	6	4	7	3	5	6	1	7	2
5	7	5	5	1	1	2	2	4	4	3	3	6	6	7

*F = Fetch Address
S = Store Address

address sequence when N=5. In this case, the fetch and store addresses may differ; note, however, that no address is stored to before its previous contents are retrieved.

4.3 OPERATING SPEED

The serial nature of the fetch and store generators obviously limits the rate at which memory accesses can be made. Nevertheless, new addresses can be generated at the rate of one address each 1.5 μsecs, far in excess of the rate at which they are required or at which the processor could process the data thus accessed.

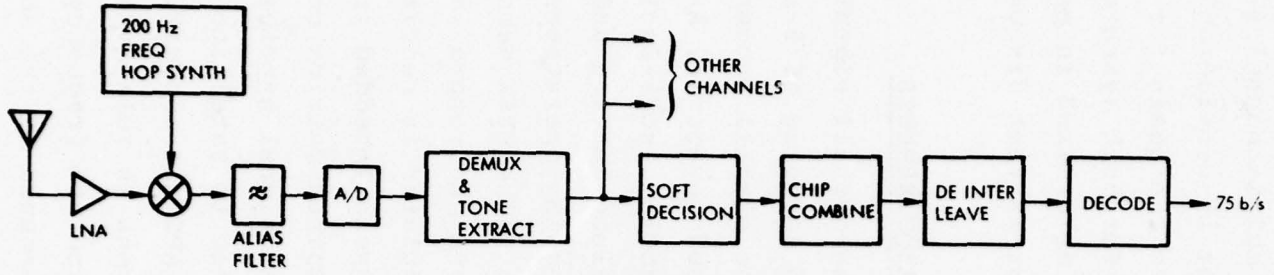
Although detailed microroutines have not been developed for the interleaving or de-interleaving algorithms, it can be readily established that the number of operations required per information bit is considerably less than the number needed in either the Decoder processor or in the Demodulator processor (cf, Section 5). The interleaving and de-interleaving functions are, of course, accomplished entirely by the address generation logic described in paragraph 4.2. The only function remaining for the de-interleaver to perform, therefore, is the chip-combining function on the de-interleaved data. The interleaver has only to perform the rate 1/2 convolutional encoding function on its input data. Neither function should seriously tax the core processor's throughput.

5.0 DEMODULATOR PROCESSOR

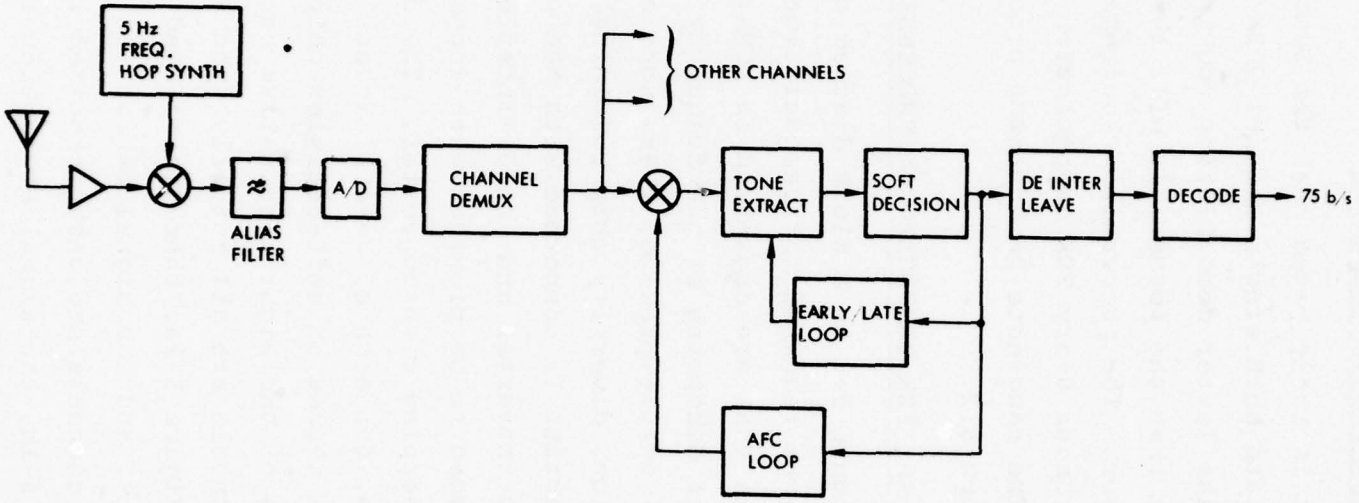
It is anticipated that the Demodulator processor will have to demodulate both single-channel QPSK and multi-channel 8-ary FSK. Since the latter demodulation requirement is considerably more complex than the former, it will serve as the basis for sizing the processor. The processing requirements for both synchronous and asynchronous 8-ary FSK demodulation are determined in paragraph 5.1. The candidate Demodulator processor is then discussed in paragraph 5.2.

5.1 8-ARY FSK DEMODULATION PROCESSING REQUIREMENTS

Figure 5-1 is a block diagram of the overall signal processing operations required for demodulation and decoding of 8-ary FSK channels that are organized in FDMA. The overall processing includes dehopping of the frequency hopped waveforms, A/D conversion, channel demultiplexing, 8-ary tone extraction, soft-decision detection, diversity chip combining, de-interleaving and decoding. This section is concerned with the processing operations between the A/D converter and the de-interleaver. The MFSK demodulator is assumed to be provided with inphase and quadrature samples from the dehopping down-converter. The demodulator is required to provide, for each of several access channels imbedded in the input data, a series of soft-decision chip-combined metrics corresponding to each of the eight alternative received signal hypothesis. If the channels are all mutually synchronized to satellite time, as in Figure 5-1a, there is no need for AFC and automatic sync recovery, and the signal extraction process is relatively simple. If the channels are unsynchronized in time and frequency, as in Figure 5-1b, the signal extraction is burdened by the need to align each channel separately in both time and frequency. Table 5-1 shows typical characteristics for 8-ary FSK channels appropriate



A) FULLY SYNCHRONIZED MFSK CHANNELS



B) UNSYNCHRONIZED CHANNELS

Figure 5-1 SIGNAL PROCESSING FOR MFSK CHANNELS

TABLE 5-1

MSFK CHANNEL PARAMETERS

CHANNEL TYPE	SYMBOL DURATION T	8-ARY TONE SPACING f	CHANNEL CARRIER SPACING Δf	HOPPING GUARDTIME INTERVAL τ	MAX. FREQUENCY MISALIGNMENT β	HOP RATE H	EFFECTIVE RAW BIT RATE S = 3/T(1-2τH)	CHIP COMBINING DIVERSITY	CODE RATE R	THROUGHPUT TRAFFIC
SYNCHRONOUS MSFK	5 ms	$\frac{1}{T}$ = 200 Hz	$\frac{8}{T}$ = 1600 Hz	0	0	200 Hz	600 b/s	4	1/2	75 b/s
UNSYNCHRONIZED MSFK	10 ms	$\frac{1}{T}$ = 100 Hz	$\frac{16}{T}$ = 1600 Hz	$\pm$25 ms	100 Hz	5 Hz	225 b/s	1	1/3	75 b/s

for both synchronized and unsynchronized channels.*

The precise time alignment in the synchronous case permits the use of frequency hop intervals equal to the waveform duration T. In this case, sequential chips are presumed to be independent with respect to CW jamming interference, and a sequence of several successive chips can be combined after soft decoding to achieve jamming diversity. For the unsynchronized case, the time misalignment of channels is such that a much slower hop rate is required to assure recovery of the waveforms of all the channels. A guard interval is required at both edges of a hop interval to accommodate the worst case channel time misalignment with the dehopping converter.

The availability of precise synchronization (or lack thereof) also imposes special considerations in channel demultiplexing and signal extraction processes. In the case of synchronized signals, the waveforms in all access channels arrive at the satellite aligned in both frequency and time. If the frequency hopping is coordinated among all access channels so that a fixed relative channel occupancy with orthogonally spaced tones pertains at the baseband output of the dehopping heterodyner, then 8-ary tone filtering for a multitude of access channels can be performed simultaneously with a single wideband FFT calculation. The processing requirements for signal separation by means of an FFT are presented in paragraph 5.1-1.

In the case of unsynchronized channel groups, the misalignment τ of the signal with respect to satellite time can be several intervals of the signal duration T and the frequency offset can be

---

*Some scaling may be needed to relate the results based on these parameters to those processing requirements actually anticipated (cf, paragraph 5.2).

as much as $\frac{1}{T}$. In the unsynchronized arrangement, timing and AFC control loops are required within the processor to acquire proper registration before tones can be correctly demodulated. This is most efficiently performed by a cascade arrangement of a down-sampling channel demultiplexer followed by separate processing in each channel to recover the associated 8-ary FSK tone structure. With this tandem processing arrangement, the timing and frequency corrections can be applied independently in each channel. The processing requirements for demultiplexing unsynchronized channels are presented in paragraph 5.1.2. The additional processing requirements associated with time and frequency control loops for the individual channels are shown in paragraph 5.1.3. Paragraph 5.1.4 describes an algorithm for soft decision decoding that is applicable for either synchronized or unsynchronized channels.

5.1.1 SIGNAL EXTRACTION FOR SYNCHRONIZED CHANNELS

In the synchronized channel arrangement, the 8-ary orthogonal tone set for a single channel occupies 8/T Hz where T is the waveform duration. If the tone assignments for M channels are arranged without interchannel guard band, the total bandwidth occupied by the channel group is 8M/T. Interchannel guard bands are not needed in a fully synchronized system because all the signal components are orthogonal. The sampling rate required to represent the multichannel composite signal in complex (I and Q) components is

$$f_s \geq 8M/T$$

At the Nyquist sampling rate, each signal interval T would contain

$$N = f_s T = 8M$$

samples. These N samples defined over the interval T can be processed by means of a discrete Fourier transform (DFT) to extract

N orthogonal frequency components with the frequency spacing between lines equal to 1/T. If the sampling rate is chosen so that N is a power of two, then the DFT can be conveniently calculated with the FFT algorithm. This sampling rate constraint requires that M be a power of two. In actual practice, some guard band is required at the band edge to accommodate the rolloff of the alias filter ahead of A/D converter, so that the edge of the band is not useful for carrying traffic. Allowing 12/5% guard band for the alias filter results in a useful channel modularity of M = 7, 14, 28, etc. when FFT processing techniques are employed.

Table 5-2 shows the processing requirements for the FFT filter as a function of the number of multiplexed channels. The operations per sample can be scaled by the channel spacing Δf = 1.6 KHz to obtain the processing speed per output channel. To account for the assumption that the alias guard band at band edge makes only 7/8 of the N_1 slots useful for traffic, the wasted slots were prorated to the other channels to obtain the per channel processing speeds. In the final columns, the per channel processing speeds are normalized to throughput in b/s by dividing the previous results by the 75 b/s channel traffic rate.

5.1.2 SIGNAL EXTRACTION FOR UNSYNCHRONIZED CHANNELS

Extraction of signal components for multichannel 8-ary FSK modulation when the channels are unsynchronized in time and frequency can be conducted in two tandem operations. In the first stage, the channels are separated by a demultiplexing filter implemented as an FFT. In the second stage, the demultiplexed subbands are separately processed by an FFT to recover the signal components of the individual channels. The second stage FFT is synchronized in time and frequency to the respective channel modulation.

TABLE 5-2

SIGNAL EXTRACTION

FOR SYNCHRONOUS 8-ARY FSK CHANNELS

NO. CHANNELS M	FFT SIZE N	PROCESSING PER INPUT SAMPLE		PROCESSING* PER INPUT CHANNEL (KOPS/SEC)		PROCESSING* PER BIT (KOPS)	
		ADD	MULT	ADD	MULT	ADD	MULT
7	64	18	12	32.4	21.6	.43	.29
14	128	21	14	37.8	25.2	.50	.34
28	256	24	16	43.2	28.8	.58	.38
56	512	27	18	48.6	32.4	.65	.43

*Indicated processing applied to chip combining/coding arrangement with 8-fold redundancy. For non-redundant transmission schemes, operations should be scaled by 1/8. Per channel processing is shown for 75 b/s channels.

A convenient channelization arrangement that is compatible with the FFT algorithm constraint of $f_s T = N$, where N is a power of two is obtained when the sampling frequency f_s, channel spacing Δf and signal duration T are related in the following manner:

$$f_s = N_1 \Delta f,$$

$$\Delta f = N_2 \frac{1}{T},$$

where N_1 and N_2 are powers of two.

With 8-ary FSK modulation in a channel, the channel occupancy is 8/T for the most compact arrangement that provides orthogonal signals. When all channels are synchronized in time and frequency, the signals in adjacent channels are also orthogonal so that no guard band is required between channels and a channel spacing $\Delta f = \frac{8}{T}$ can be used. However, lack of synchronization among channels destroys orthogonality so that the channels must be separated with adequate guard band to avoid interchannel spillover. Increasing the channel spacing to $\frac{16}{T}$ provides a frequency separation between extremal tone frequencies in adjacent channels of $\frac{7}{T}$. This corresponds to interchannel spillover of about -27 dB for rectangular pulse envelopes of duration T which produce a corresponding power spectrum of the form

$$H(f) = \left| \frac{\sin \pi f T}{\pi f T} \right|^2 .$$

If the channel spacing is $\Delta f = \frac{16}{T}$ and the sampling frequency $f_s = N_1 \Delta f$, with N_1 a power of two, an FFT demultiplex filter can be implemented that will provide N_1 resolution lines. Not all of these lines can be used for access channels because of the need for guard bands at the band edges to accommodate the rolloff of the alias filter ahead of the A/D converter. Allowing a 12.5% guard

band for this filter reduces the number of useful channels to $\frac{7}{8}N$, so that channel modularity of M = 7, 14, 28, etc. channels can be conveniently obtained by FFT processing.

5.1.2.1 Demultiplex Filter Processing

Figure 5-2 shows the filter characteristics appropriate for demultiplexing a multichannel group of unsynchronized 8-ary FSK signals. In the figure the channel spacing is $\Delta f = \frac{16}{T}$, the sampling rate of the complex input signal is $f_s = N_1 \Delta f$, and the main lobe spectral width (between nulls) of each demultiplex channel is Δf. In order to avoid channel spillover among the demultiplex channels, the first and subsequent sidelobes of the subband filters are required to be better than 30 dB down.

A demultiplex filter that meets these requirements can be achieved using a $4N_1$-point FFT with raised cosine amplitude weighting to suppress resolution filter sidelobes. (In fact, with raised cosine weighting the largest sidelobes will be nearly 40 dB down.) A $4N_1$-point transform is inefficient, however, because this process computes the spectral components for $4N_1$ subbands and only every fourth such component corresponds to an access channel center frequency. Also, the output sampling rate for each of the $4N_1$ subbands is only equal to $\frac{f_s}{4N_1} = \frac{\Delta f}{4}$, whereas the mainlobe width of the filter is Δf, so that a four-fold overlap of the FFT process would be required to achieve the Nyquist sampling rate for the subsequent tone extraction process in the channel processors.

The required demultiplex processing can be performed much more efficiently by decimating the $4N_1$-point FFT to eliminate those butterfly operations that do not affect the N_1 spectral components of interest. A processing arrangement to achieve that is shown in Figure 5-3. In this scheme, an N_1-point FFT algorithm is cascaded

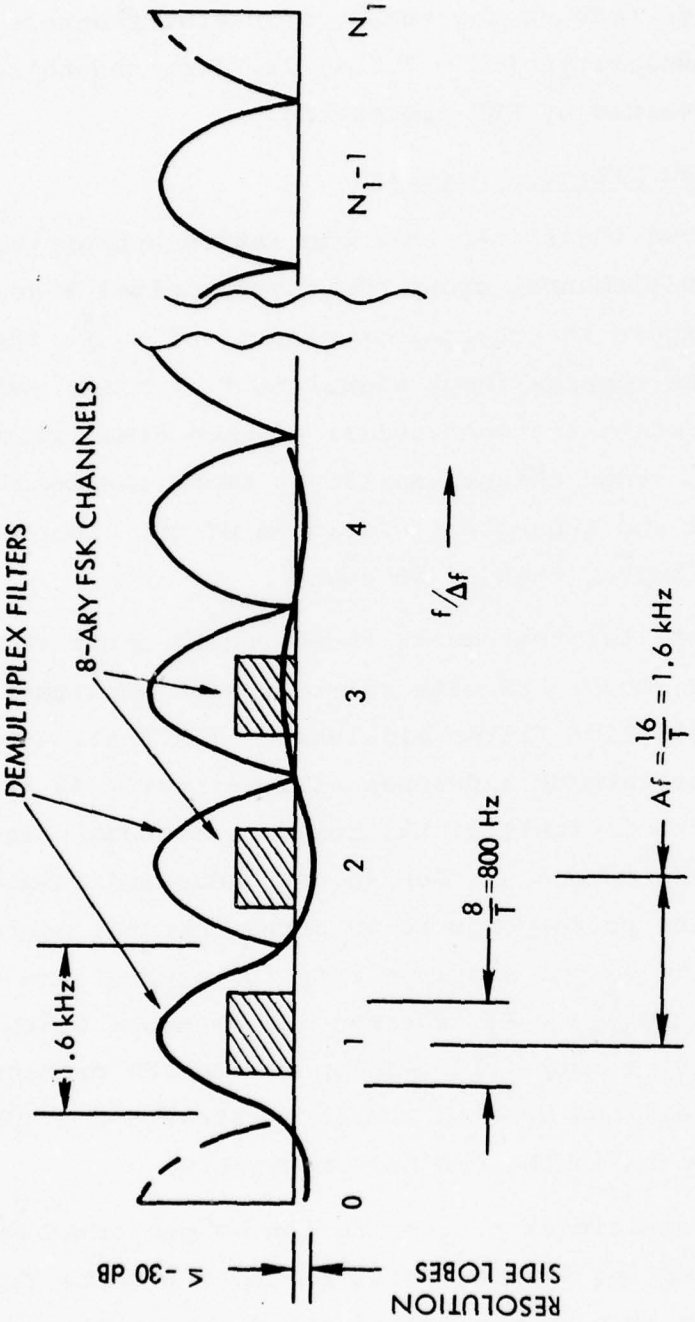


Figure 5-2 DEMULTIPLEX FILTER CHARACTERISTICS

with a preprocessor. The combined process computes every fourth line of a $4N_1$-point FFT with fourfold overlap and raised cosine amplitude weighting. The N_1-point FFT computes the upper quarter of the rightmost rank of the conventional FFT array (i.e., every fourth frequency component) and the preprocessor computes all the butterfly operations to the left that thread into the upper right quarter of the array. In the first stage of the preprocessor, raised-cosine amplitude weighting is used to suppress the resolution sidelobes to a level near -40 dB. Four separate preprocessor paths are used in the amplitude weighting to fold the input samples into four blocks of data containing $2N_1$ time-domain samples that represent the even spectral lines of $4N_1$-point FFT's with raised-cosine weighting and 75% time overlap. The overlap switches alternately route a sequence of $2N_1$ samples from a pair of first-stage paths to the second stage of the preprocessor. The second-stage process is carried out in two parallel paths in the same manner as the first stage, except amplitude weighting is omitted. The outputs of the second-stage processor paths are folded time domain samples in blocks of N_1 samples that represent every fourth spectral line of $4N_1$-point FFT's, again with raised-cosine weighting and 75% time overlap. The multiplexing switch alternately routes a block of N_1 samples to the N_1-point FFT from the two second-stage preprocessors. As a result, the time multiplexed output from the N_1-point FFT provides output samples that represent the spectral content of each of the multiplex filters shown in Figure 5-2. With the fourfold time overlapped transforms, the output sample rate per channel is equal to the subchannel Nyquist rate Δf. As shown by the sidelobe levels in Figure 5-2, the above channel sampling rate is adequate to reduce alias foldover to a level well below -30 dB. As stated earlier, the raised cosine amplitude weighting will provide sidelobe levels near -40 dB.

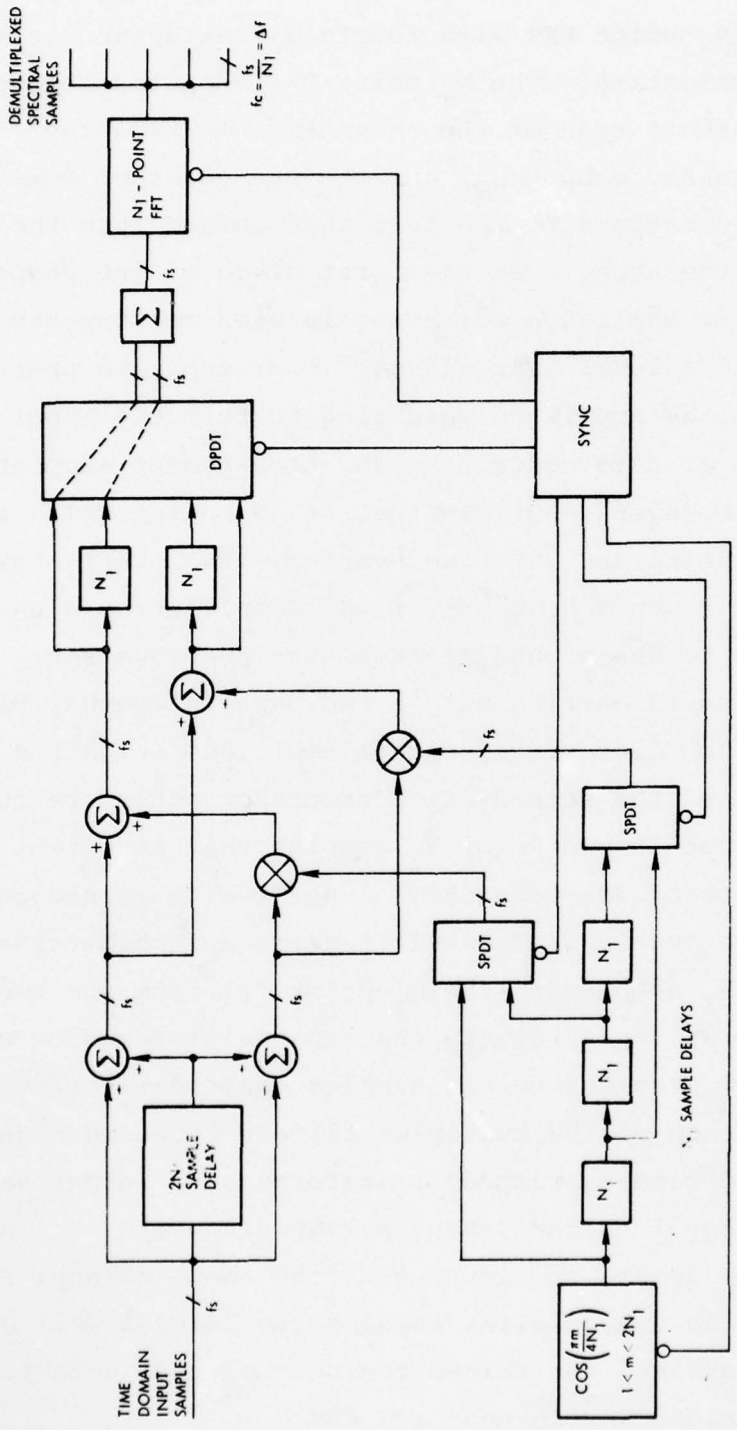


Figure 5-3 DEMULTIPLEX PROCESSOR

The processing operations in the preprocessor consist of five complex additions and two half-complex multiplications (or equivalently 10 real additions and four real multiplications) per complex input sample, independent of N_1. The processing operations in the cascaded N_1-point FFT are three $\log_2 N_1$ real additions and two $\log_2 N_1$ real multiplications per complex input sample.

Table 5-3 shows the processing requirements for the demultipled FFT filter as a function of the number of multiplexed channels. The operations per sample can be scaled by the channel spacing Δf = 1.6 KHz to obtain the processing speed per demultiplex output channel. To account for the assumption that the alias guard band at band edge makes only 7/8 of the N_1 slots useful for traffic, the wasted slots were prorated to the other channels to obtain the per channel processing speeds. In the final columns, the per channel processing speeds are normalized to throughput in b/s by dividing the previous results by the 75 b/s channel traffic rate.

5.1.2.2 Channel Signalling-Tone Processor

The channel signalling-tone processor is provided with a sample sequence representing the signal and noise components out of the corresponding demultiplex filter. The sampling rate is equal to the channel spacing Δf, and corresponds to the spectral width of the main lobe of the multiplex filter as shown in Figure 5-2. Prior to performing the FFT to extract the frequency components, the input sample sequence is heterodyned by the AFC frequency correction to center the tone frequencies in the resolution cells of the FFT. The AFC process is described in paragraph 5.1.2.3. The FFT operation is synchronized to the waveform through the operation of an early-late synchronizing loop that aligns the FFT channel window to the waveform envelope. The early-late synchronizing process is described in Section 5.1.2.4.

TABLE 5-3

DEMULTIPLEX FILTER

FOR UNSYNCHRONIZED 8-ARY FSK CHANNELS

NO. CHANNELS M	FFT SIZE N_1	PROCESSING PER INPUT SAMPLE		PROCESSING* PER INPUT CHANNEL (KOPS/SEC)		PROCESSING* PER BIT (KOPS)	
		ADD	MULT	ADD	MULT	ADD	MULT
7	8	19	10	34.2	18.	.45	.24
14	16	22	12	39.6	21.6	.52	.29
28	32	25	14	45.	25.2	.60	.34
56	64	28	16	50.4	28.8	.67	.38

*Indicated processing applies for a coding scheme with 3-fold redundancy and 25% guard interval per frequency hop. Results should be scaled by 1/4 to obtain values for a non-redundant transmission scheme without guard time. Per channel processing is shown for 75 b/s channels.

The AFC corrected samples occur at a sampling rate of $\Delta f = 1.6$ KHz, and the signalling tones are spaced by $\frac{1}{T} = \frac{\Delta f}{16} = 100$ Hz. Therefore, a $N_2 = 16$-point FFT is required to recover the eight tone frequencies of interest. A 16-point FFT requires $\frac{N}{2} \text{Log}_2 N = 32$ butterflies.

The eight signal components are obtained from FFT resolution cells 4 through 11. From Figure 5-2, it is observed that the spectral components corresponding to the different tone frequencies are weighted differently owing to the bandpass shape of the demultiplex filter. Therefore, the last stage of the FFT must provide amplitude weighting to equalize the amplitude of the tone components (i.e., noise whitening). The required amplitude scaling can be programmed into the multiplication coefficients in the final FFT butterflies so that no additional arithmetic operations are needed. The required processing operations for the 16-point FFT are shown in Table 5-4.

5.1.2.3 Synchronization of Demultiplexer Output Channels

The demultiplexer provides output samples at the channel Nyquist rate of $\Delta f = \frac{16}{T} = 1.6$ KHz. These samples must be heterodyned by a complex sinusoid to compensate for the frequency offset of the channel, and thereby center the 8-ary FSK tone frequencies in the resolution cells of the FFT tone filter. Likewise, the FFT tone filter must be synchronized to the pulse envelopes of the channel signalling. Both of these synchronization processes require tracking loops to sense the offset error and to adjust the processor to null the error. In conventional channel usage schemes, a fixed-format sync preamble is transmitted ahead of the test to provide an interval for the time and frequency tracking loops to acquire the signal and resolve ambiguities in time and frequency. If the preamble is chosen so that a known pair of tone frequencies are

TABLE 5-4

MFSK SIGNAL EXTRACTION

FOR DEMULTIPLEXED ASYNCHRONOUS CHANNELS

FFT SIZE N	PROCESSING PER INPUT SAMPLE		PROCESSING* PER INPUT CHANNEL (KOPS/SEC)		PROCESSING* PER BIT (KOPS)	
	ADD	MULT	ADD	MULT	ADD	MULT
16	12	8	19.2	12.8	.256	.171

*Indicated processing applies for a coding scheme with 3-fold redundancy redundancy and 25% guard interval for frequency hop. Results should be scaled by 1/4 to obtain values for a non-redundancy transmission scheme without guard time. Per channel processing is shown for 75 b/s channels.

transmitted alternately for sync, then a special matched-filter processor can be devised to achieve sync. Once sync is achieved, the timing and frequency connections can be held for the duration of the message.

Another approach to synchronization can be devised that does not require the transmission of a preamble. This approach is of great interest for use in LPI report-back channels where minimum transmission time is advantageous. In this scheme, it is necessary to buffer the demultiplexer output to save the message until the appropriate timing and frequency offsets can be derived, at which time the buffered data can be synchronously demodulated. The required delay in the buffer will generally be somewhat greater than the duration of the preamble that is replaced owing to the fact that the message structure will generally not be of optimum form for rapid sync recovery.

Figure 5-4 shows a block diagram for a synchronization system that can be used for channels that do not contain message preambles. As shown, the first block of data appearing in the demultiplexer output is processed twice in the FFT. The first FFT operation is used to recover time and frequency synchronization. At the end of the first block of L symbols (i.e., 16L samples), the input switch of the FFT is activated so that the delayed (by L symbols) demultiplexer output samples are routed to the FFT processor for a second pass. These samples are synchronously demodulated with the appropriate time and frequency adjustments as derived from the first pass. The buffered data continues to be processed until the demultiplexer output is again connected to the FFT processor to initiate the synchronization cycle for the next message in the channel.

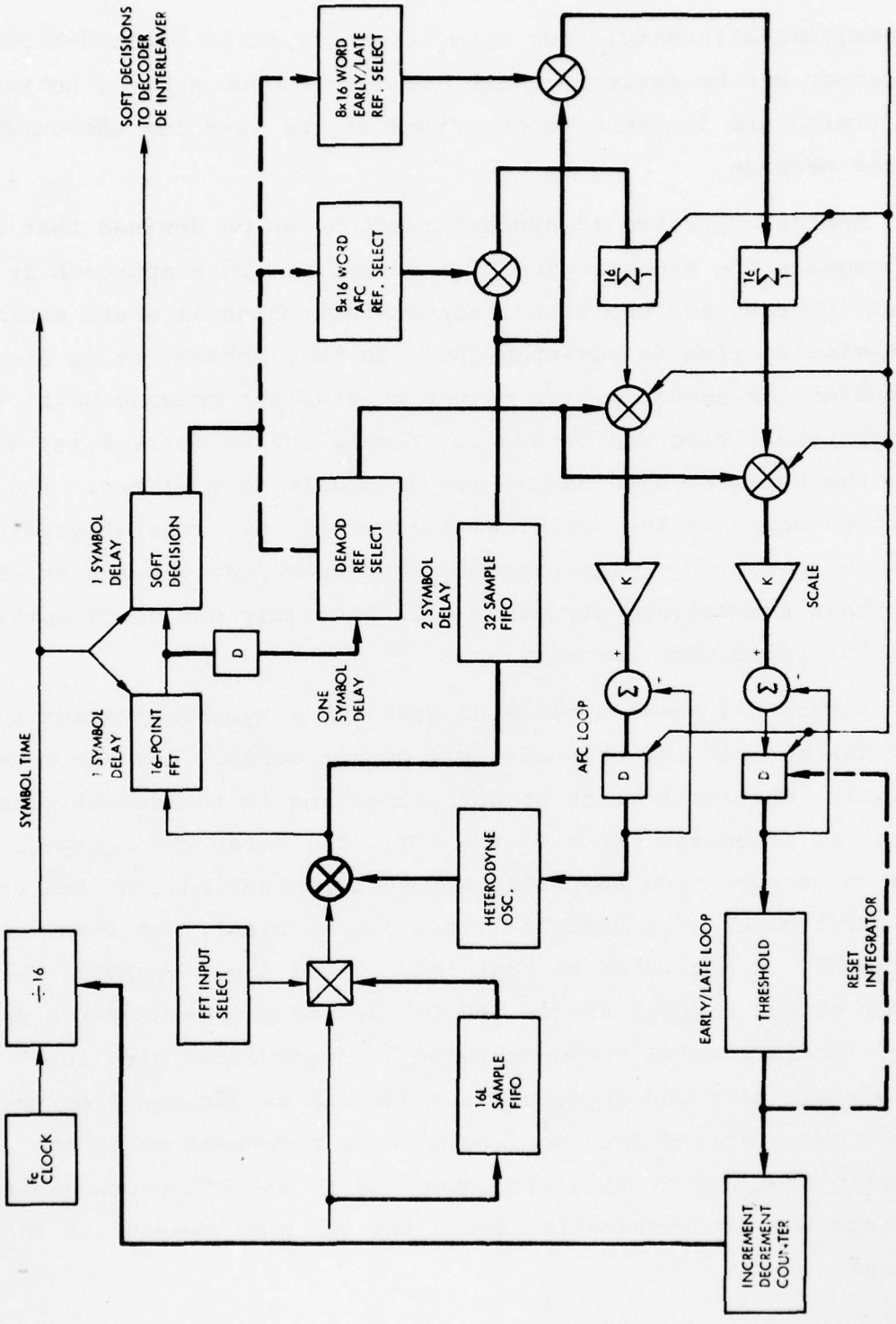


Figure 5-4 AFC AND EARLY/LATE SYNCHRONIZATION

5.1.2.3.1 AFC Control Loop

The AFC error sensor is implemented with a 16-point cross correlation process as shown in Figure 5-4. The heterodyned samples that are applied to the FFT are delayed by two symbol intervals (32 input samples) to permit the soft-decision decoder to determine the most probable transmitted symbol in the 8-ary symbol alphabet. This decision is used to select one of eight cross-correlation reference signals to apply to the 16 input samples that were used by the FFT to derive the decision variable.

The eight cross-correlation reference signals, $4 \leq m \leq 11$, are the complex sample sequences

$$X_{n_m} = \exp\left(-j \frac{\pi nm}{8}\right) \left\{2j \sin\left(\frac{\pi n}{16}\right)\right\}.$$

for $0 \leq n \leq 15$, $4 \leq m \leq 11$.

The cross correlation of the appropriate reference signal with the sample sequence into the FFT produces a "discriminator" response with a null at the center of the frequency cell. The cross-correlation output is synchronously demodulated by the signal component as derived by the FFT to produce a frequency error indication that is linear with offset error. The processing requirements for the AFC cross correlator correspond to one complex multiply and one complex add per input sample. After the complex cross-correlation metric is computed, the result is coherently demodulated with a half-complex multiply performed once per symbol interval T using the signal sample previously extracted with the FFT as the demodulation reference. The real part of the demodulated error signal is scaled by the servo loop gain and smoothed in a recursive integrator (lag filter) to derive the offset frequency for

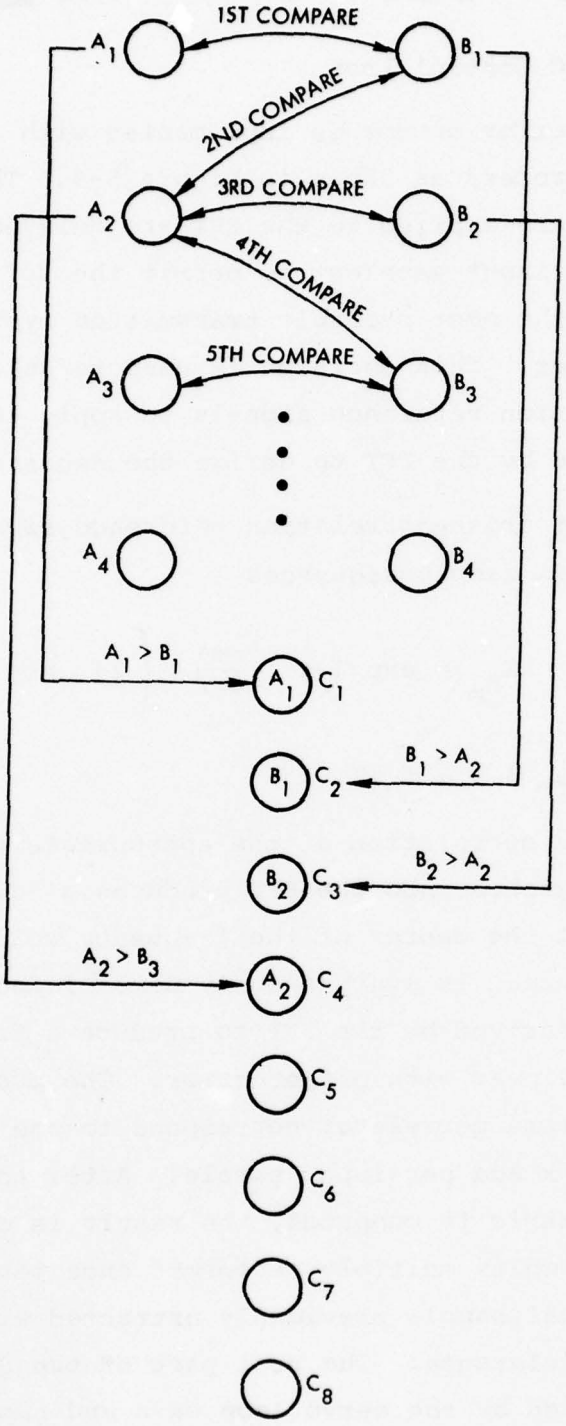


Figure 5-5 MERGING OF RANKED CHAINS

the heterodyned oscillator. This offset frequency is interpreted as the phase slope per sample interval in the AFC heterodyner oscillator. The heterodyning oscillator computes a new phase angle once per sample interval in a recursive integrator using the relationship $\phi_n = \phi_{n-1} + \Delta\phi$, where $\Delta\phi$ is the phase slope per input sample interval as obtained from the loop filter. The computed phase can be truncated to about 5 significant bits to obtain an address for accessing a small sine/cosine table to obtain each complex sample of the heterodyning sequence. The heterodyning sequence does not require great precision; however, the phase integrator requires sufficient precision to provide an AFC frequency step increment that is narrow compared to the resolution bandwidth of the FFT that follows the heterodyning mixer. Referenced to the sample rate f_c = 1.6 KHz, the use of eight-bit precision in the phase integrator would provide an AFC step size of 6.6Hz or 1/16 of the FFT resolution width.

5.1.2.3.2 Early/Late Control Loop

The early/late control loop operates in essentially the same manner as described for the AFC loop with the following exceptions:

(1) The correlation reference signals are defined over the set of alternative signal tones $4 \le m \le 11$ by

$$y_{nm} = \exp\left(-j\,\frac{\pi nm}{8}\right) A_n,\ 0 \le n \le 15$$

$$A_n = \begin{cases} -1, & 0 \le n \le 7 \\ +1, & 8 \le n \le 15 \end{cases}$$

(2) The heterodyning oscillator is replaced by an error threshold and a timebase increment/decrement process.

Because there are 16 input samples in the signal interval T, the early/late sync loop can align system timing to within 6.6% of

the ideal sync timing. A timing offset of 6.6% introduces a maximum loss in carrier to noise ratio of about 0.5 dB, and a negligible intersymbol spillover. Better accuracy could be achieved by using an interpolation filter, but the performance gain would be slight and the processing requirements would be nearly doubled.

As in the case of the AFC loop, the appropriate correlation reference is selected by the soft decision processor and the output from the cross correlation process is synchronously demodulated using the previously derived signal component from the FFT as the demodulation reference. After the demodulated early/late error is smoothed in the integrator, the resultant is applied to a threshold detector. When the error magnitude exceeds the threshold, the time base counter (divide by 16) is incremented or decremented as appropriate to reduce the timing offset error. At the same time, the integrator is cleared and the loop operates on the new timing offset until the integrator output next exceeds the threshold. Under steady-state conditions, the loop timing oscillates between the discrete timing offsets that bracket the ideal timing. During acquisition, the pull-in behavior will closely approximate that of a linear first order servo, except that there will tend to be an overshoot of one sample interval.

5.1.2.3.3 Processing Summary for AFC/SYNC

Table 5-5 shows the processing requirements for AFC and time synchronization broken out by the individual processing blocks of Figure 5-4. Each of the two cross-correlation operations corresponds to one full-complex multiply and one complex add per input sample (i.e., 4 real multiplications and 4 real additions). The synchronous demodulation of the error signals requires half (the real part) of a full complex multiply performed once per symbol (i.e., every 16 input samples). The integrator and loop gain scaling together

TABLE 5-5

AFC/SYNC PROCESSING

OPERATION	PROCESSING PER INPUT SAMPLE		PROCESSING* PER INPUT CHANNEL S/SEC)		PROCESSING* PER B/S (OPS)	
	ADD	MULT	ADD	MULT	ADD	MULT
16-Point Correlations (AFC & Sync)	4	8	6.4	12.8	85.3	160.6
Synchronous Demodulate (AFC & Sync)	0.25	0.125	0.4	0.2	5.3	2.6
Loop Integrator (AFC & Sync)	0.125	0.125	0.2	0.2	2.6	2.6
Heterodyne Oscillator (AFC)	1	--	1.6	--	21.3	--
Early/Late Threshold Compare (Sync)	0.5	--	0.8	--	10.6	--
Mixer (AFC)	2	4	3.2	6.4	42.7	85.3

*Indicated processing applies for a coding scheme with 3-fold redundancy and 25% guard interval per frequency hop. Results should be scaled by 1/4 to obtain values for a non-redundant transmission scheme without guard time. Per channel processing is shown for 75 b/s channels.

require one real multiply and one real addition per symbol. The heterodyner oscillator requires one full complex multiply per input sample (4 real multiplies and 2 real additions).

5.1.2.4 Soft-Decision Processing

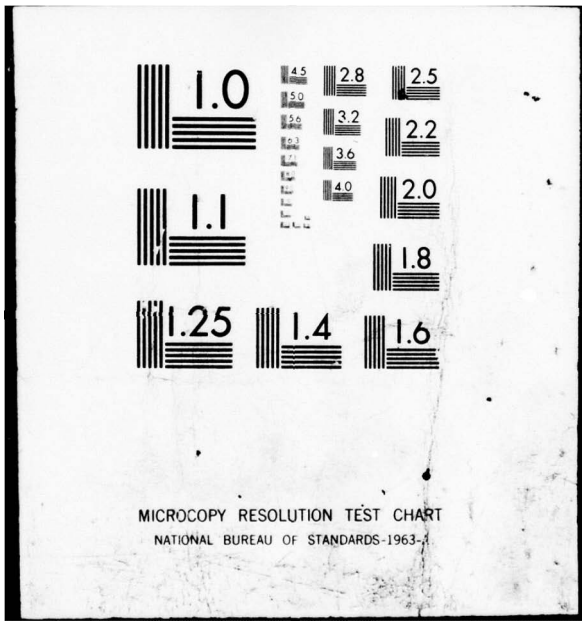
The soft-decision processor computes decision metrics for each of the eight alternative signal hypothesis on the basis of the received energy in the eight tone locations. There are a wide variety of ways that the envelope metrics can be mapped into decision metrics, and it is well known that the optimum transformation is that which produces the log-likelihood ratio. Unfortunately, the log-likelihood transformation is a function of the channel noise statistics and cannot be rigorously defined for channels in which the noise characteristics are not known, as in jamming or self interference environments. Under these conditions, a transformation is required that is robust under impulsive noise conditions and still provides near optimum performance for white Gaussian noise channels.

One such transformation that has been proven very effective in this regard is envelope rank listing. In this process, the envelopes associated with each of the eight signal alternatives are compared, and each of the alternatives is assigned a number from 0 to 7, depending on the relative rank of its envelope as compared with all other alternatives. Thus, the largest envelope is assigned a value 7 and the smallest a value of 0. Each of the numbers 0-7 is assigned to an alternative, and no two alternatives receive the same number. The process as described can result in $8! = 40,320$ different assignment combinations. It is impractical to directly test all of these possibilities for each soft decision, so that a more systematic method of arriving at the result with minimum arithmetic operations is required.

AD-A067 693 RAYTHEON CO SUDBURY MASS EQUIPMENT DIV F/G 9/2
FAULT TOLERANT SPACEBORNE COMPUTER STUDY. PREPROCESSOR DESIGN.(U)
JAN 79 J J STIFFLER F04701-77-C-0050
UNCLASSIFIED ER79-4005 SAMSO-TR-79-27 NL

2 OF 2
AD
A067693

END
DATE FILMED
6 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

One efficient algorithm for accomplishing the rank listing is based on systematic merging of two rank listed chains so that the merged chain is also rank listed. The process is illustrated in Figures 5-5 and 5-6. The process starts by comparing the leading members of each chain. The contender with the greatest magnitude (say A_1) is selected as the highest ranked member C_1 of merged chain C. With this procedure, two chains of length 4 require a total of 6 passes, and the final chain of length 8 requires 7 passes. The whole process then requires 17 passes which is equivalent to 170 instruction executions. Allowing 30 executions for initialization and envelope calculation (e.g., by truncation the real and imaginary terms to four bits each and using a table look-up) gives a total of 200 equivalent additions per soft decision.

Table 5-6 shows the processing requirements for soft-decision detection for synchronized and unsynchronized MFSK channels. The instructions executed are a mix of arithmetic and conditional jump operations, but no multiplications are involved so the processing load would be equivalent to the indicated number of additions.

Chip combining is accomplished by summing the soft-decision metrics over a number of chip intervals corresponding to the chip locations where the same symbol is repeated. If the soft-decision metrics were truly log-likelihood ratios, then the combined output would correspond to the log-likelihood ratio for the combined chips. The rank-list soft-decision process produces 3-bit numbers that are an approximation to the true log-likelihood ratio.

Combining the soft-decision metrics from four chips produces numbers that range over the values $0 \leq U \leq 28$, so that the output of the chip combiner must be represented by a five-bit number to avoid further loss of information content contained in the envelope

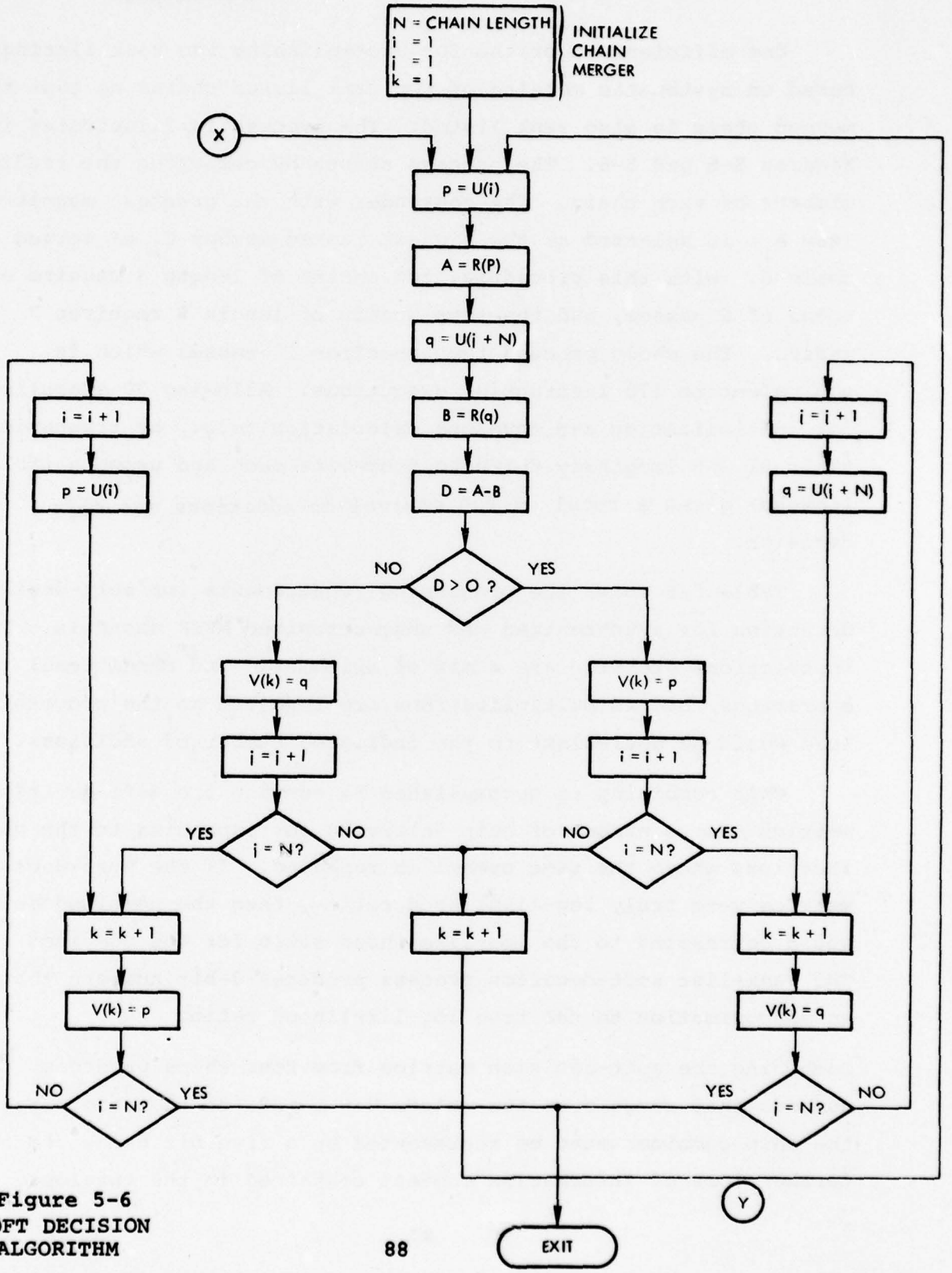


Figure 5-6
SOFT DECISION
ALGORITHM

TABLE 5-6

MFSK SOFT DECISION DETECTION

CHANNEL TYPE	INSTRUCTIONS PER SYMBOL	SYMBOL RATE	PROCESSING OPERATIONS PER CHANNEL (KOPS/SEC)	PROCESSING OPERATIONS PER BIT (KOPS)
Fully Synchronized MFSK	200	200*	50*	0.53*
Unsynchronized MFSK	200	100**	20**	0.27**

*Indicated processing applies to chip combining/coding arrangement with 8-fold redundancy. For non-redundant transmission schemes operations should be scaled by 1/8. Per channel processing is shown for 75 b/s channels.

**Indicated processing applies for a coding scheme with 3-fold redundancy and 25% guard interval per frequency hop. Results should be sealed by 1/4 to obtain values for a non-redundant transmission scheme without guard time. Per channel processing is shown for 75 b/s channels.

amplitudes in the four chips composing the combined output. The chip combining process requires only 8 additions per chip interval, which is a negligible contribution to the overall demodulation process.

5.2 DEMODULATOR PROCESSOR IMPLEMENTATION

The Demodulator processor postulated for the WBSP can be regarded as two core processors, configured to run in parallel, combined with a 512-word by 22-bit sample-point memory. (The reason for this 22-bit memory width is explained below.) In addition, each core processor is provided with an 8 x 8 parallel multiplier. (It is anticipated that a currently existing multiplier chip can be used for this purpose.) The core processor pair, however, is less complex than two independent processors for two reasons: 1) They share the same bus interface. 2) They perform essentially the same operations at the same time; thus, most fields of the control RAM can be shared. Specifically, the 40-bit width control RAM needed for a single processor is expanded to 56 bits for the dual processor.

The sample-point memory is organized into 512 words, each word divided into a 9-bit real and a 9-bit imaginary part to which is appended an overall parity bit. Three spare bits and a bit rippler are added for fault tolerance.

The 9-bit word length was chosen on the basis of a quantization noise simulation described in Appendix A. It was found in that simulation that the signal-to-quantization-noise ratio associated with FFT processing ranged between 33 and 40 dB under a wide variety of conditions when the data were truncated to 8 bits, a ratio felt to be entirely acceptable. The quantization noise was kept to that level by dividing every data value by two after each FFT iteration if, and only if, at least one butterfly operation

resulted in an overflow. (Note that a butterfly operation can at most double the magnitude of its inputs.) This in effect requires the data to be represented in a floating-point format with a single-bit exponent used to indicate whether or not an overflow occurred during the previous iteration. (An alternative method would be to access all previously processed data and divide each data point by two as soon as the first overflow occurred during any iteration, an obviously time-consuming procedure.) Accordingly, 9 bits of memory are allowed for each data point, with the 9th bit used to store the associated overflow bit.

The memory control chip described in Section 4 can be used to control the sample-point memory as well. The address shuffler is readily modified to implement the much simpler addressing sequence required for an FFT and most of the rest of the logic in that chip is applicable as well. In addition, a "normalizer" can easily be integrated onto that chip, so that the divide-by-two operation can occur automatically, as the data are passed to the processor, whenever an overflow occurred during the previous FFT iteration.

The Demodulator processor operates in two phases when demodulating MFSK data. (The much simpler QPSK cross-link data demodulation process can be implemented using a single-phase procedure.) During phase 1, the processor's sample-point memory is loaded with the data samples taken during the current symbol interval. At the same time, soft-decision outputs are calculated on the basis of the previously transformed samples and transferred to the Interleaver/De-interleaver processor. The data stored during phase 1 are then processed in accordance with the algorithm described in paragraph 5.2 (except for the soft-decision portion of that algorithm) during phase 2. The dual-processor accepts both the real and imaginary portions of two sample-point memory outputs,

performs the appropriate butterfly operation, and sends the results back to the sample-point memory to be stored for the next iteration.

Two Demodulator processors are required for each FDM MFSK channel, one implementing phase 1 while the other implements phase 2. The two processors switch roles after each symbol interval. To verify that a pair of such processors can indeed implement the algorithms described in paragraph 5.2, recall that, when $N_1 = 16$, asynchronous demodulation (the more complex case) requires a total of 71,400 equivalent additions and 54,000 multiplications per second per channel to implement all of the phase 2 operations. (Many fewer operations are required during phase 1.) Since these numbers are based on fourteen channels (and are not substantially reduced if only twelve channels are present), approximately 10^6 equivalent adds and $3/4 \times 10^6$ multipliers are needed each second to demodulate one asynchronous FDM MFSK signal. If the symbol interval is halved (to 5 msec), these rates are essentially doubled. Since the processor contains hardware multipliers, and additions and multiplications both require the same amount of time (one micro-cycle), the total number of operations needed per second at the 5 msec symbol rate is thus 3.5×10^6. This number must be increased by a factor of approximately two to account for the time needed to access and store the data, for interation loop overhead, etc. Thus, the Demodulator processor must be able to execute microinstructions at a 7×10^6 instruction/second rate if it is to handle an asynchronous MFSK FDM channel in the manner described here. Since a core processor can execute microinstructions at a 6.67×10^6 instruction/second rate and since a Demodulator processor consists of two core processors operating in parallel, it can clearly implement the required processing algorithm with a comfortable margin.

6.0 RELIABILITY, POWER, WEIGHT, VOLUME AND THROUGHPUT SUMMARY

6.1 WBSP SIZING

Table 6-1 lists the functional assignments of the various Demodulator, Interleaver/De-interleaver, and Encoder/Decoder processors needed to configure the WBSP for anticipated SSS applications. The number and types of LSI devices used to implement each of these three processor types are also summarized in Table 6-1.

6.2 FTSC SIZING

The FTSC is modular in design so that the number of memory blocks and I/O ports can be tailored to each specific application. As indicated in Figure 2-1, two I/O ports, one parallel and one serial port, are postulated for the SSS configuration. It is estimated that three active 4096-word memory blocks are required to store all the program, constants and variable data needed for the SSS mission. This estimate was based on a detailed examination of the memory requirement associated with each of the tasks to be performed. The results of this examination are summarized in Table 6-2.

Also shown in Table 6-2 is the maximum percentage of the FTSC's total throughput required to perform each of the identified SSS tasks. As can be seen, the presently identified tasks leave the FTSC with a throughput margin adequate for future growth.

6.3 RELIABILITY ANALYSIS

Reliability analyses were conducted for both the WBSP and the FTSC. For purposes of these analyses, an LSI device hazard rate of 10^{-7} failures per hour was postulated and a dormancy factor (ratio of active to dormant hazard rates) of 1.1 was used where applicable.

TABLE 6-1

PROCESSOR COMPLEXITY AND UTILIZATION SUMMARY

a) Encoder/Decoder Processor (Core Processor)

COMPOSITION

Element	No. of Devices
8-Bit RALU	2
Data RAM	2
Control RAM	10
I/O Buffer and Control RAM Sequencer	1
Bus Interface	6
TOTAL	21

UTILIZATION

Dual-3 Decoder	1
Feedback or Reed-Solomon Decoder	1
Cover and Queen's Code Encoder	1
TOTAL NEEDED	3

b) Interlever/De-interleaver Processor

COMPOSITION

Element	No. of Devices
Core Processor	21
Intl. Mem.*	84
Memory Control	1
Bit Rippler	2
TOTAL	108

UTILIZATION

Report Back De-Intl.	4
FWD. Link De-Intl.	3
EHF. Link De-Intl.	1
Interleaver	1
TOTAL NEEDED	9

*3096 words of 24 information bits, 1 parity bit and 3 spare bits each: 84 1024 x 1 RAM chips.

TABLE 6-1 (Cont.)

c) Demodulator Processor

COMPOSITION

Element	No. of Devices
Core Processor	21
2nd RALU	2
2nd I/O Buffer and Control Sequencer	1
2nd Data RAM	2
Dual Multipliers	2
Expanded Control RAM	4
Sample Point Memory*	11
Memory Control	1
Bit Rippler	1
TOTAL	45

UTILIZATION

Report Back Demod	2
FWD and EHF Demod	2
Cross-Link Demod	1
TOTAL NEEDED	5

*512 words of 18 information bits, 1 parity bit and 3 spare bits each.

TABLE 6-2

FTSC LOADING FOR SSS

	Program/ Constant Storage	Dynamic Storage	Instructions/ Iteration		Exec. Time Per Iteration (m sec)	Iteration Rate/Sec (When Active)	% Duty Cycle
			Short	Long			
NAVIGATION							
Orbit Element Comp	250	70	810	210	4.95	1	.51
Perturbation Eqns	500	60	4500	1980	37.26	1/1800	.002
Coord. Conversions	100	40	450	150	3.15	1	.33
Time Update	20	5	20	0	0.05	200	1.20
GUIDANCE							
Orbit Insertion	300	20	240	100	1.92	1	.19
Evasion/Recovery	300	20	240	100	1.92	1	.19
Station Keeping	200	20	100	60	1.02	1	.10
ATTITUDE INFORMATION UPDATE							
Strapdown Algorithms	160	60	420	50	1.86	20	3.72
Star Selection	250	20	750	1500	20.25	1/200	.01
Star Tracker Processing	100	20	330	140	2.67	1/200	.001
Filter	600	125	900	340	6.78	1/200	.003
Attitude Initialization	250	100	600	100	-	-	-
ATTITUDE CONTROL							
Major Reorient	100	10	120	60	1.08	20	2.16
Rotate to Nadir	50	10	90	40	0.75	1	.08
Device Pointing	200	30	120	60	1.08	20	2.16
Propulsion Steering/ Countdown	50	10	70	40	0.69	100	6.90
MATH SUBROUTINES	300	20	N/A	N/A			
SUBTOTALS - GUID/NAV/CONTROL	3730	640					17.56
FTSC OPERATING SYSTEM							
Executive							
Scheduler	200	200	100	-	.3	200	6.0
Input/Output Handling	800	300	100	-	.3	200	6.0
Other	200	40	N/A	N/A	N/A	N/A	N/A
Fault Monitor Tests	1100	300	10000	-	30	1/3600	.001
Recovery (Fault)	400	-	N/A	N/A	N/A	N/A	N/A
WBSP COMMUNICATIONS							
Routing Data Management	400	300	1200	-	3.6	1	.36
Fault Monitoring	500	60	1000	-	3.0	1	.3
Fault Recovery	1600	-	500	-	1.5	1	.15
Antenna Scan Control	200	50	150	50	1.05	1	.105
COMMAND DECODING	100	-	30	-	.09	5	.045
TELEMETRY	200	-	100	30	1.32	20	2.64
TOTALS	9430	1850					33.16%

The predicted 7-year reliability of the proposed WBSP configuration is .9654. The basis for this prediction is shown in Table 6-3 which lists the complexities and redundancies for each of the elements comprising the WBSP as well as the resulting 7-year reliability of each element. The complexity of each element is indicated in terms of the number of equivalent LSI devices. (The hazard rate is therefore equal to the complexity multiplied by 10^{-7}.) The complexity of an element may differ slightly from its actual device count due to interconnection complexity factors and due to the fact that a portion of the failures occurring at the bus interfaces appear as bus (byte) failures rather than failures in the element associated with that interface.

The 7-year reliability of the FTSC is predicted to be .9592. This prediction is based on the reliability model used for all FTSC reliability calculations but with the following exceptions: six, rather than four, CPUs were used in order to increase the mission duration from five to seven years; the second DMA port was eliminated, since no need for it has been established.

6.4 RELIABILITY, POWER, WEIGHT AND VOLUME SUMMARY

The estimated power requirements of the various WBSP elements are listed in Table 6-3. The estimated power consumption of the FTSC in the configuration proposed here is 21.2 watts.

Weight and volume estimates for the WBSP were made under the assumption that it would use the packaging techniques developed for the FTSC. The smaller (4.3" x 10.0") of the two FTSC modules was assumed for all WBSP elements; the number of modules needed to accommodate each of the WBSP elements under these conditions is also indicated in Table 6-3. The FTSC weight and volume estimates were extracted from results derived during that program.

TABLE 6-3

WBSP RELIABILITY AND POWER SUMMARY

Element	Complexity	# Avail./ # Needed	Reliability (7 yrs.)	# Modules Required	Power Required
Bus Controller	34.5	2/1	.9946	1	1.9
Bus 1	2.63	3/2	.9992	-	-
Bus 2	3.13	3/2	.9989	-	-
Bus 3	2.75	3/2	.9992	-	-
Bus 4	2.0	3/2	.9996	-	-
Demodulator	44.5	9/5	.9943	4 1/2	3.8
Deinterleaver	108.0	14/9	.9922	14	8.3
Decoder	20.5	6/3	.9976	1 1/2	2.5
A/D Conv.	6.0	$(2/1)^5$	.9935	1	3.0
Modulator	4.6	$(2/1)^4$	.9969	0.9	1.2
Timing Generator	3.5	2/1	.9995	0.1	0.4
Power Supply	9.8	2/1	.9966	-	3.7
		TOTAL REL. ---	.9627		
		TOTAL # MODULES ---		23	
		TOTAL POWER REQUIREMENTS ---			24.8 Watts

The results of these calculations are summarized in Table 6-4. The first two rows in that table give parameters for the WBSP and the FTSC when the two units are packaged individually. The third row gives the same parameters for the WBSP and FTSC packaged as one unit. If they are to be packaged individually, the reliability and power totals remain unchanged but the weight and volume of the two individual units are just the sums of the values shown in the first and second rows.

TABLE 6-4

RELIABILITY, POWER, WEIGHT, AND VOLUME SUMMARY

	7 yr. Reliability	Power (watts)	Weight (lbs)	Volume (ft^3)
WBSP	.9627	24.8	36.0	0.52
FTSC	.9592	21.2	44.8	0.65
TOTAL	.9234	46.0	67.7	1.05

7.0 CONCLUSIONS AND RECOMMENDATIONS

The results of the WBSP study thus far have demonstrated that the fault tolerance techniques developed for the FTSC can also be used effectively to configure a highly reliable communications processor of the sort needed for SSS. The same structure that gives the signal processor a high reliability-improvement-to-redundancy ratio (i.e., its multiple, reassignable processor configuration) also makes it unusually versatile. Coding or modulation techniques can be changed, frame lengths can be modified, and channels can be added or deleted without changing the WBSP hardware design; generally, such changes can be accommodated by making software changes and by adding or removing some number of processing elements.

The cost of developing the proposed WBSP should be much lower than normal for a system of this complexity since so much of the work is already being done for the FTSC. All but three of the LSI devices identified for the WBSP are being developed for the FTSC. These devices will be subjected to extensive reliability and hardness testing as part of the FTSC program, thus providing the data base needed for the WBSP as well. Moreover, all of the difficult design problems associated with a fault-tolerant system (e.g., fault detection and isolation) have already been addressed in the FTSC design. The solutions developed there (e.g., the bus structure) are being used intact in the WBSP; thus, the WBSP development effort can concentrate on the more tractable problem of designing the individual processors themselves.

The proposed structure for the WBSP has been defined during the course of the present study, and reasonably detailed designs have been developed for each of the three processor types needed for its implementation. Considerable work remains to be done, however, to

demonstrate the concept throughly. To this end, the following two-phase effort is proposed:

Phase I DESIGN AND VERIFICATION

Task 1: Define remaining microroutines:

a) Synchronous FDM MFSK demodulator
b) Asynchronous FDM MFSK demodulator
c) TDM QPSK demodulator
d) De-interleaver
e) Interleaver
f) Reed-Solomon Decoder

These microroutines should be defined in a manner similar to that described in Section 3; register utilization and scratch-pad memory partitioning must also be determined.

Task 2: Complete detailed LSI chip design:

a) I/O Buffer and Control RAM Sequencer
b) Memory Control

Both these chips have been defined functionally; this functional description must now be translated into detailed logic designs.

Task 3: Design Bus Controller

The Bus Controller design should be relatively straightforward compared to the processor designs comprising the bulk of the effort in the present study. Nevertheless, the Bus Controller design must obviously be completed if the WBSP is to function as a unit. It would appear that the Memory Control chip to be designed in Task 2 could easily be used to control the Bus Controller memory as well; this possibility should be investigated.

Task 4: Design A/D Converter Module

The A/D Converter Module postulated for the WBSP consists of a relatively low-speed (< 100 KHz) converter coupled with bus interface logic. A monolithic A/D converter apparently suitable for this purpose has already been identified so this design should pose no serious problems. An investigation should be made into the advisability of preceding the converter with a prescaler, so that the signal gain can be changed under demodulator control.

Task 5: Design Modulator Module

The Modulator Module postulated for the WBSP simply translates each digital symbol received over the WBSP bus into one of four (QPSK) or eight (MFSK) discrete levels defining the subcarrier phase or frequency appropriate for each FDM or TDM channel. The design of such a device is obviously straightforward. It is, of course, possible to design the modulator to generate the subcarriers directly, and hence to reduce the complexity of the subsequent analog hardware at the cost of increased modulator complexity. Either special-purpose or general-purpose digital modulators could be used in this capacity; in the latter case, a modulator pooled-spares configuration with an analog multiplexer between the modulators and the r.f. system becomes an attractive possibility. The desirability of such design modifications should be examined.

Task 6: Design Verification

It is advisable, when dealing with a system as complex as the WBSP, to verify that the design is correct before committing it to hardware. One method for doing this is to simulate the design on a computer. Such a simulation was in fact written for the basic (Encoder/Decoder) processor during the present study using a

higher-order simulator language called CDL (computer design language). The product of this effort is described in Appendix B. It is proposed that this simulation be extended to include the Demodulator and Interleaver/De-interleaver processors as well and that extensive runs be made to verify that all of the processing algorithms perform as required.

Phase II FABRICATION AND TEST

Task 7: Brassboard Fabrication

Once the design has been completed and its correctness verified, it is proposed that a brassboard be constructed and used to test and demonstrate the WBSP concept. Initially, only one of each of the major WBSP elements (Bus Controller, A/D converter, Demodulator, De-interleaver, Decoder and Modulator) would be needed. This "single-string" brassboard would be sufficient to demonstrate the performance characteristics of the WBSP as a communications processor; all of the processing algorithms could be demonstrated individually and in the appropriate combinations. It would be desirable, however, eventually to add the multiple elements needed to demonstrate a full-scale WBSP configuration with all the various signal processing tasks taking place simultaneously. Such a full-scale brassboard could then be coupled with the FTSC (the existing FTSC brassboard is recommended for this purpose) in order to test and demonstrate the integrated WBSP-FTSC system.

It should be noted that a WBSP brassboard could be constructed quite economically by again taking advantage of the FTSC program. An engineering test model (ETM) of the FTSC is currently under development. The ETM will be constructed from logic cards implemented with existing SSI and MSI logic. These cards, however, will be functionally identical with the LSI devices being developed for

the FTSC and will be interchangeable with them. Since the WBSP is designed to use these same LSI devices, most of the WBSP brassboard can be constructed simply by making extra copies of the cards already being fabricated for the ETM. Such a procedure will obviously reduce significantly the time and expense needed to construct a WBSP brassboard.

Task 8: Test and Demonstration

The purpose of this task is to carry out the performance tests referred to in the previous task description. The single-string brassboard would be used to obtain communication system performance parameters (e.g., bit-error rate vs. signal-to-noise and signal-to-jammer ratios for all modulation and coding schemes of interest.) The full-scale brassboard would be used primarily to test and demonstrate the WBSP's ability to handle multiple channels, the FTSC's routing and fault-monitoring capability and the integrated system's tolerance to faults.

APPENDIX A

FFT PROCESSING AND QUANTIZATION NOISE

A.1 INTRODUCTION

Processor complexity and signal processing speed are directly related to the length of the registers. Due to the inherently non-linear nature of fixed-point arithmetic implemented with finite-length registers, however, the resulting quantization errors are difficult to estimate analytically. Although a linear quantization noise model can be used, the results obtained from it are often unreliable due to the fact that quantization noise depends heavily on the dynamic nature of the input signal. In general, it is not adequate to assume that quantization noise is independent of the input signal, and as a result of the cross-correlation between the input signal and the quantization noise, predication of the induced noise level is a difficult task.

Accordingly, a computer simulation program was written so that the effect of performing FFTs using finite-length registers could be evaluated. Noise levels were measured by applying random input signals and parametric curves were obtained relating the noise and signal levels.

The FFT algorithm assumed for purposes of this discussion is the conventional "decimation-in-frequency" FFT performed using b-bit data words. It is further assumed that sealing is accomplished dynamically: that is, when an overflow occurs in an iteration, the entire data base is scaled by 1/2 and the iteration is continued at the point at which the overflow occurred.

The assumed quantization function is given by the following expression

$$x_q = \frac{\text{Integer Part of}\left[\frac{x}{x_{max}} \cdot (2^{b-1} - 1)\right]}{2^{b-1} - 1} \; x_{max}$$

where x is the voltage to be quantized and x_{max} is the maximum A/D range, taken arbitrarily to be 10 volts for the FFT arrays and 1 for the sine and cosine tables.

A.2 SIMULATION RESULTS

A.2.1 BANDLIMITED GAUSSIAN NOISE PLUS TONE

In this simulation, a 32-point FFT was taken of a single tone signal plus Gaussian noise. The tone (in frequency bin 4) had a 5-volt peak amplitude and the noise had an rms value of 1 volt in all frequency bins except bins 13 through 18. These bins were left empty in order to enable spectral leakage measurements. The results of this simulation are shown in Figures A-1 through A-6 for data word lengths b varying from 16 bits to 8 bits. A "perfect" (48-bit) transform was also made for reference purposes (Figure A-1). These results are summarized in Figure A-7.

A.2.2 FDM SIGNALS

Several simulations were made to determine interchannel interference effects caused by data word truncation. These simulations modeled m 8-ary FSK channels with m = 8, 16, 32, 64. The active tone in each channel and its phase (any one of 8 equally-spaced values) were chosen using a pseudo-random sequence. The spectrum produced by taking an N = 8m-point FFT of the composite signal was then determined. The results for m = 64 are shown in Figure A-8 through A-13 for various data word lengths b. These results along with similar results for m = 8, 16 and 32 are summarized in Figure A-14 which shows the signal-to-quantization-noise ratio as a function of N and b. For this simulation, the active tones were each assumed to have an amplitude of 10/64 volts and the quantization noise was averaged over all frequency bins. Figure A-15 shows the same results when the composite signal consists of equal-

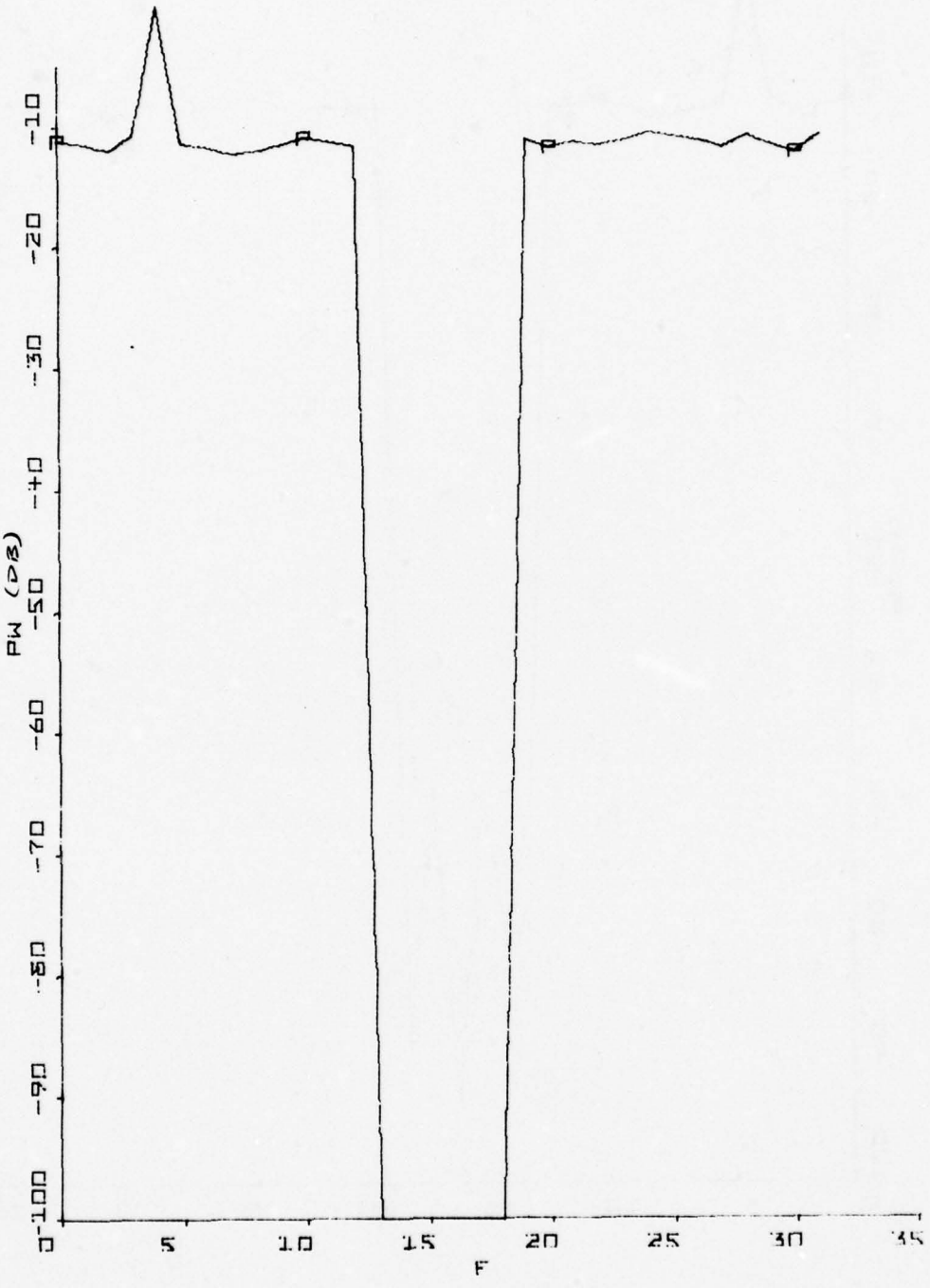


Figure A-1 _TONE PLUS GAUSSIAN NOISE SPECTRUM (b=48)

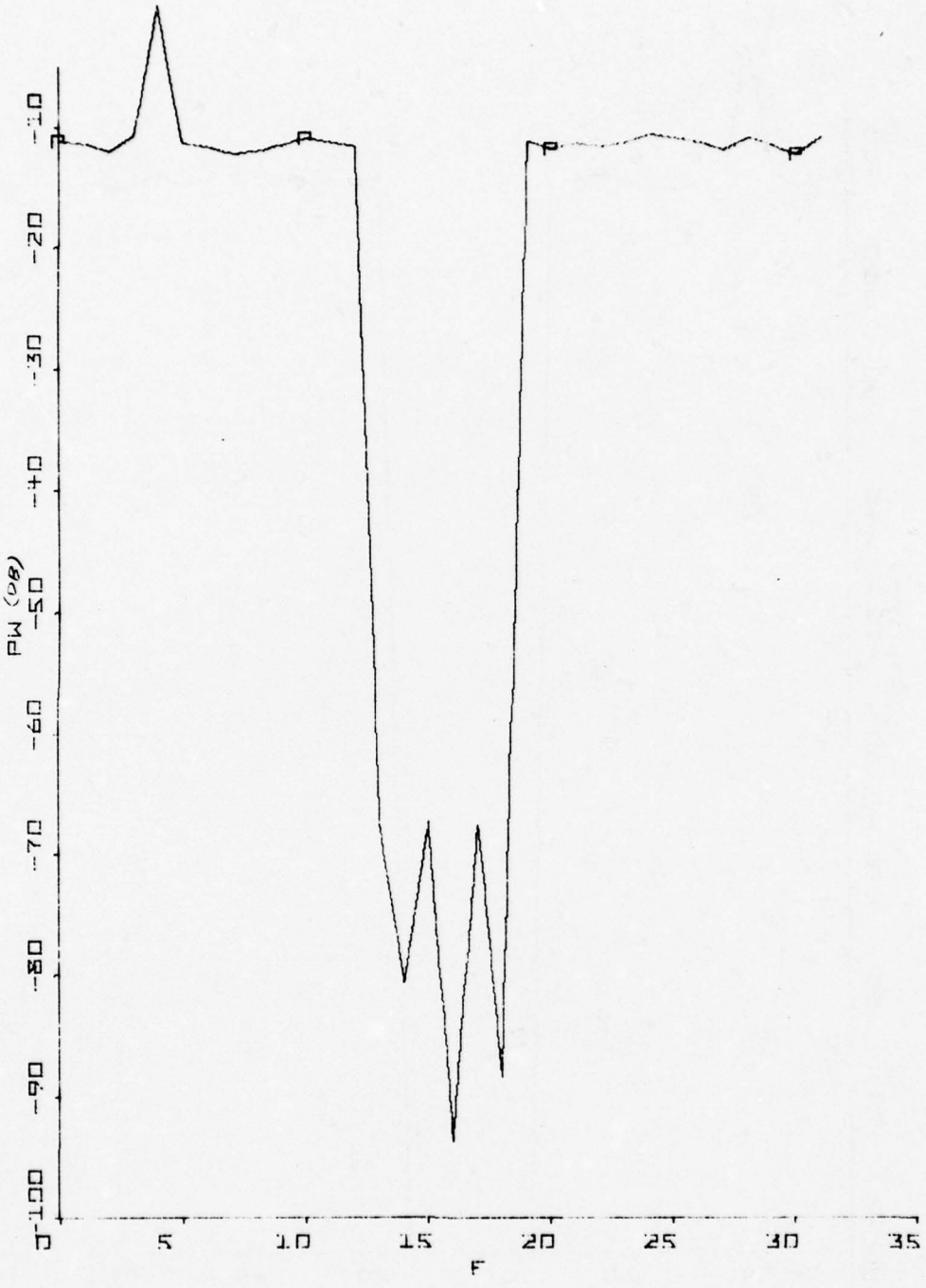


Figure A-2 TONE PLUS GAUSSIAN NOISE SPECTRUM (b=16)

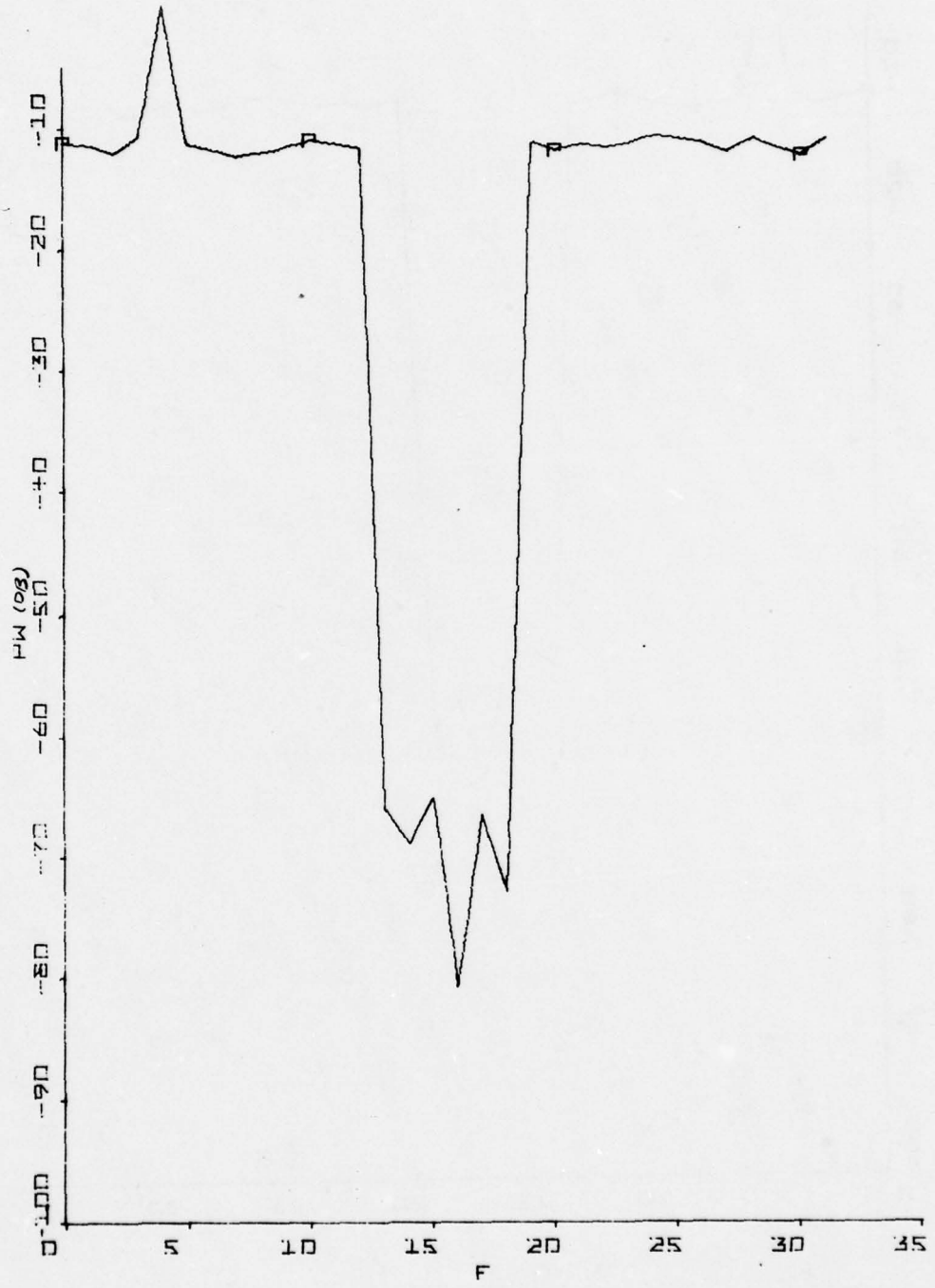


Figure A-3 TONE PLUS GAUSSIAN NOISE SPECTRUM (b=14)

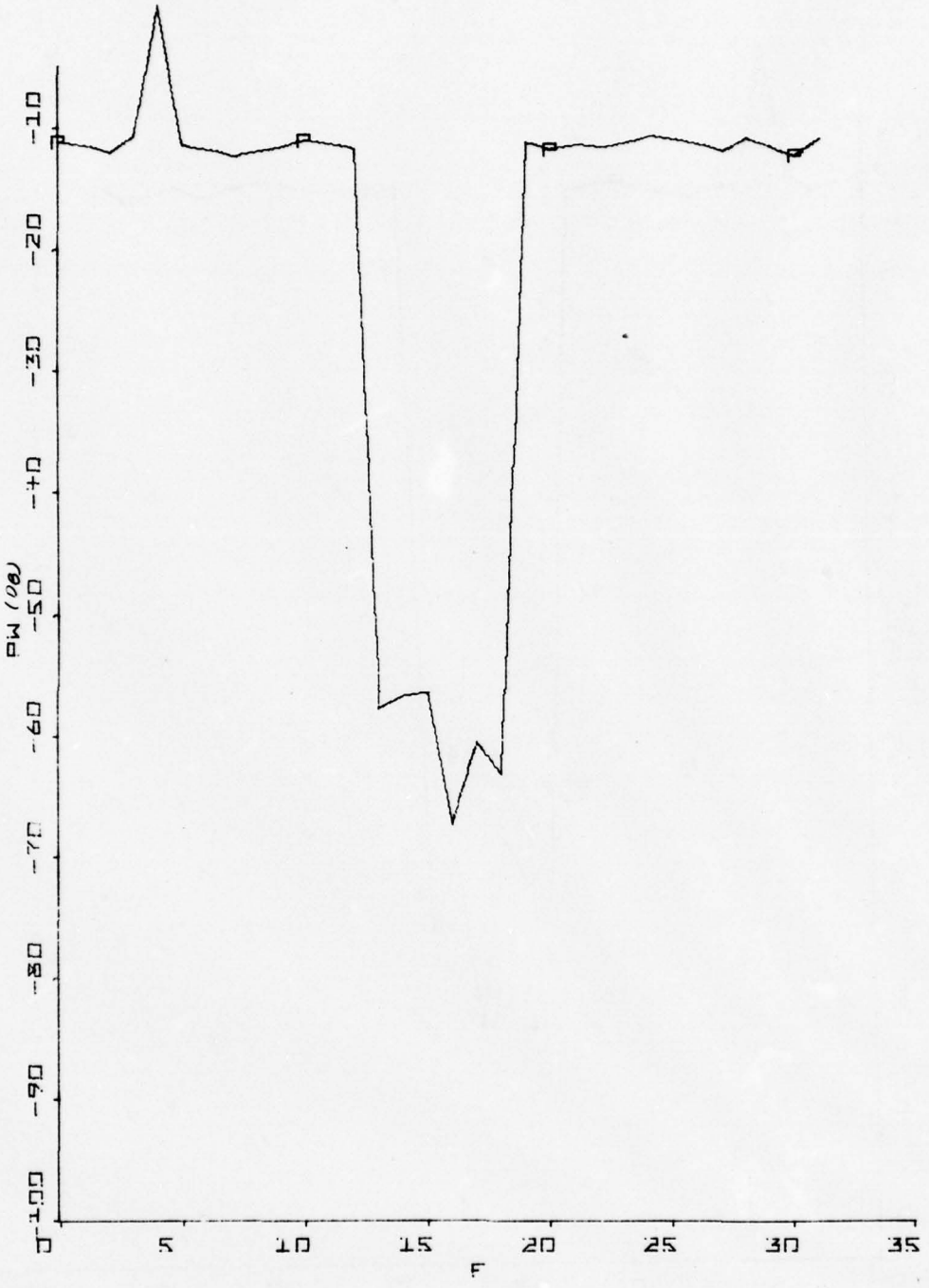


Figure A-4 TONE PLUS GAUSSIAN NOISE SPECTRUM (b=12)

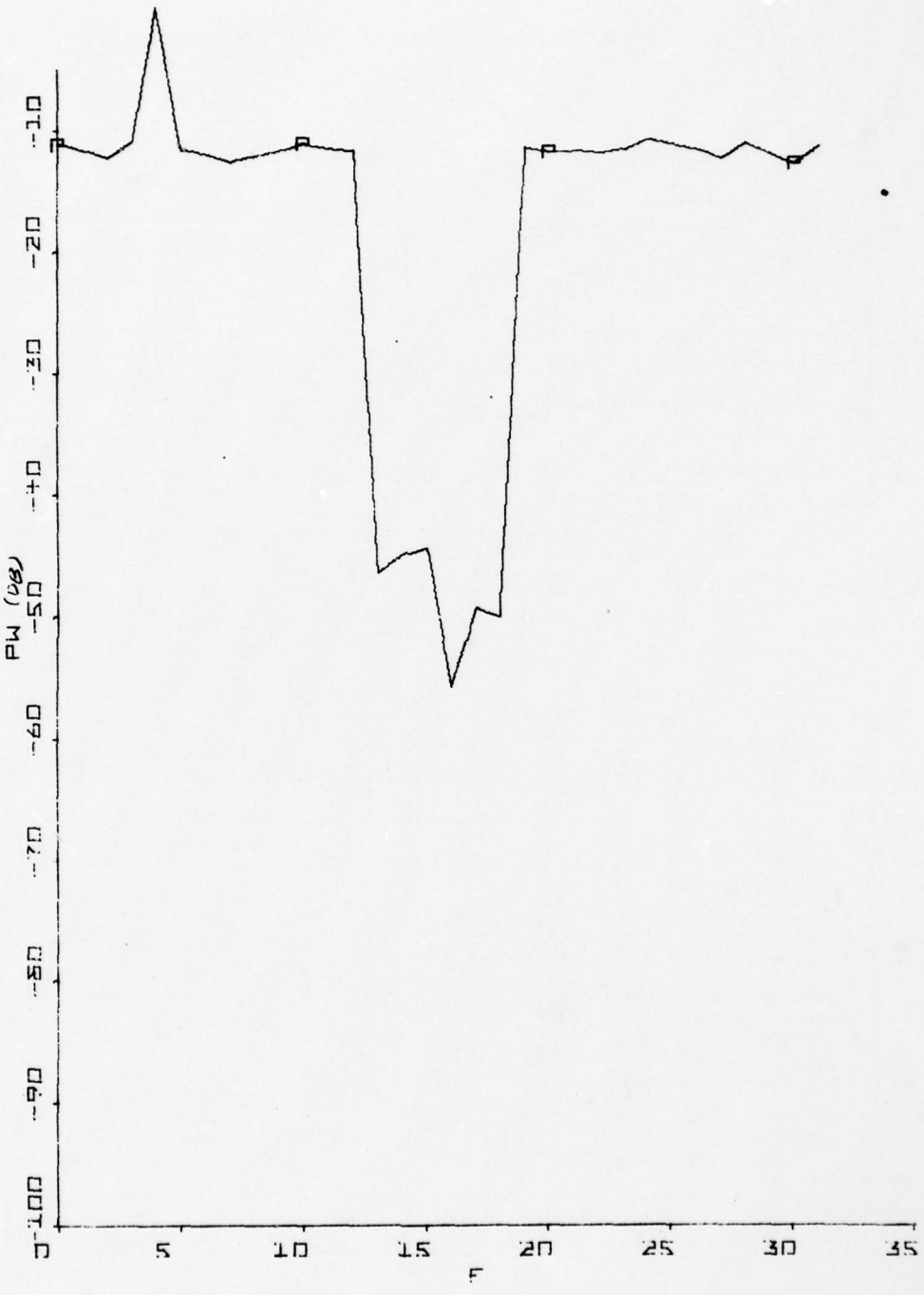


Figure A-5 TONE PLUS GAUSSIAN NOISE SPECTRUM (b=10)

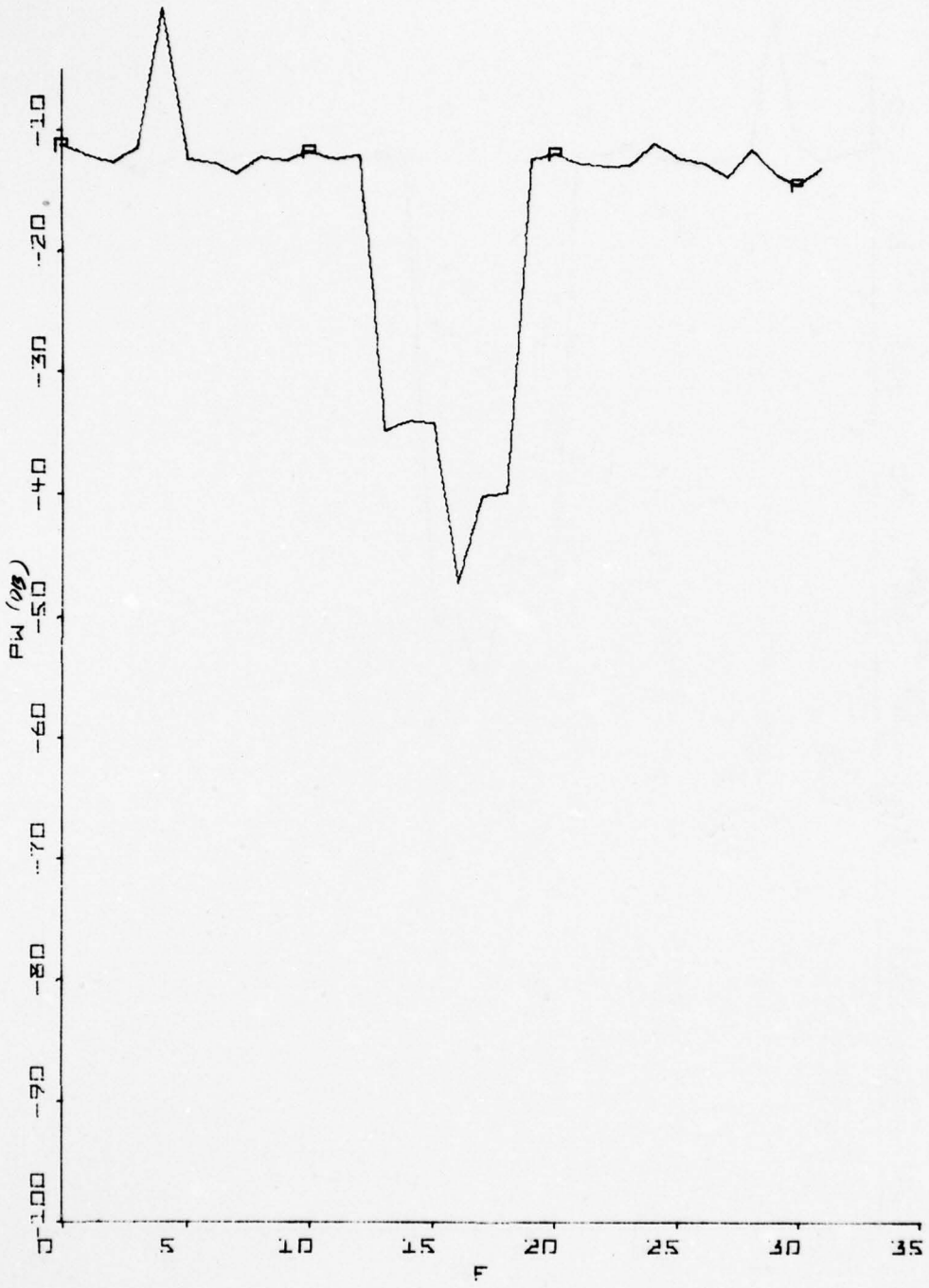


Figure A-6 TONE PLUS GAUSSIAN NOISE SPECTRUM (b=8)

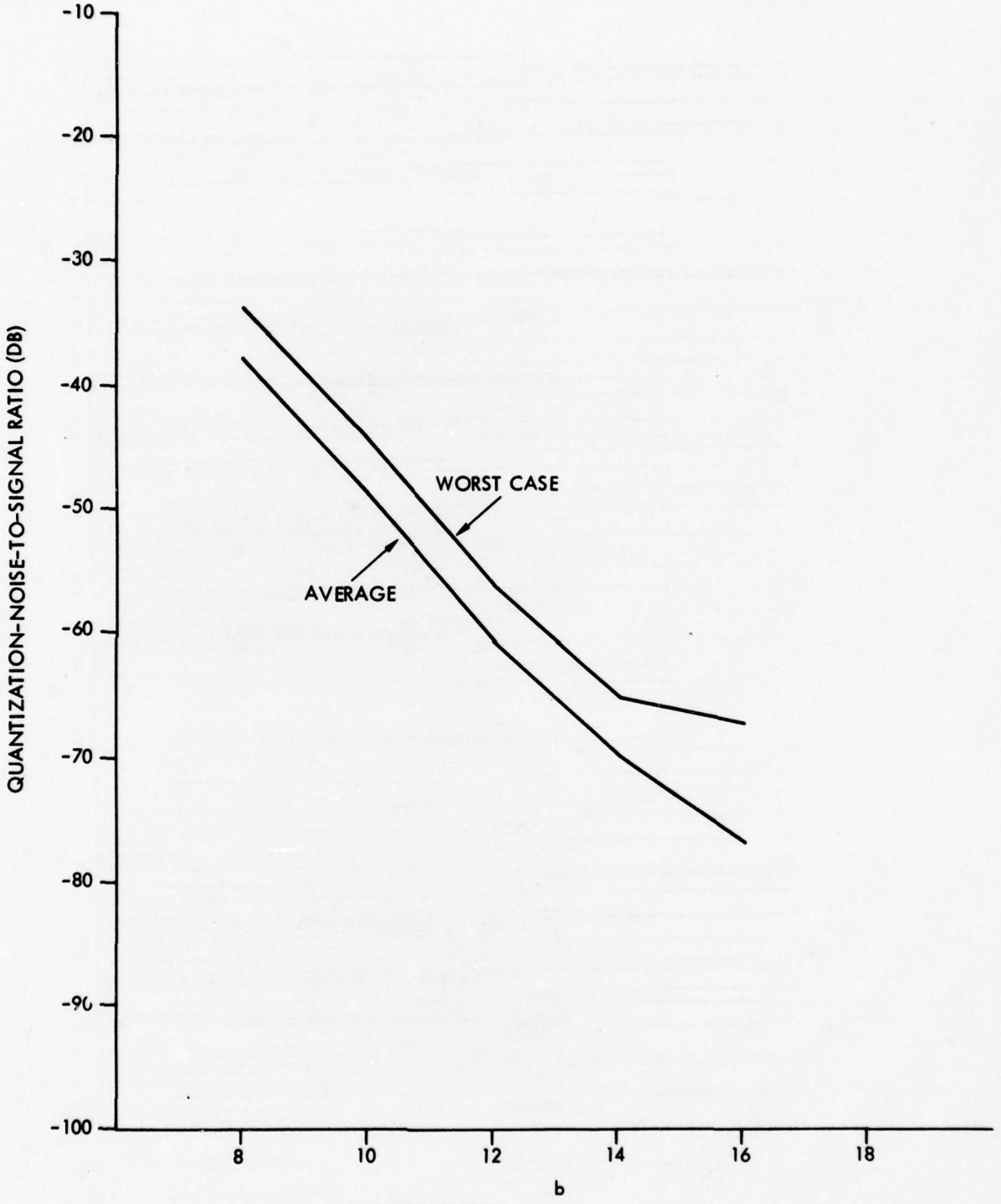


Figure A-7
QUANTIZATION NOISE VS. WORD LENGTH b (TONE PLUS GAUSSIAN NOISE)

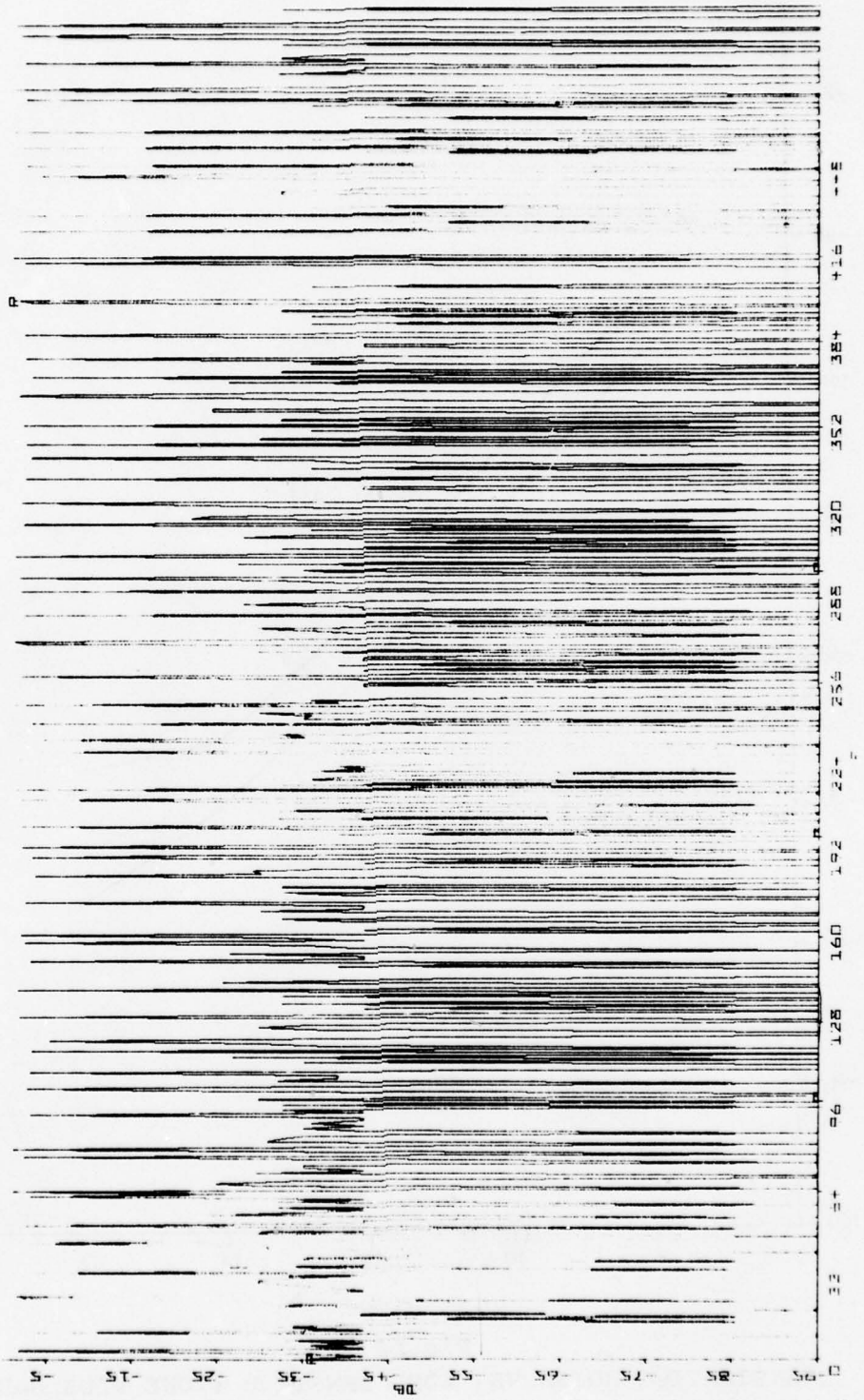


Figure A-8 FDM SPECTRUM (N= 512, m=64, b=8)

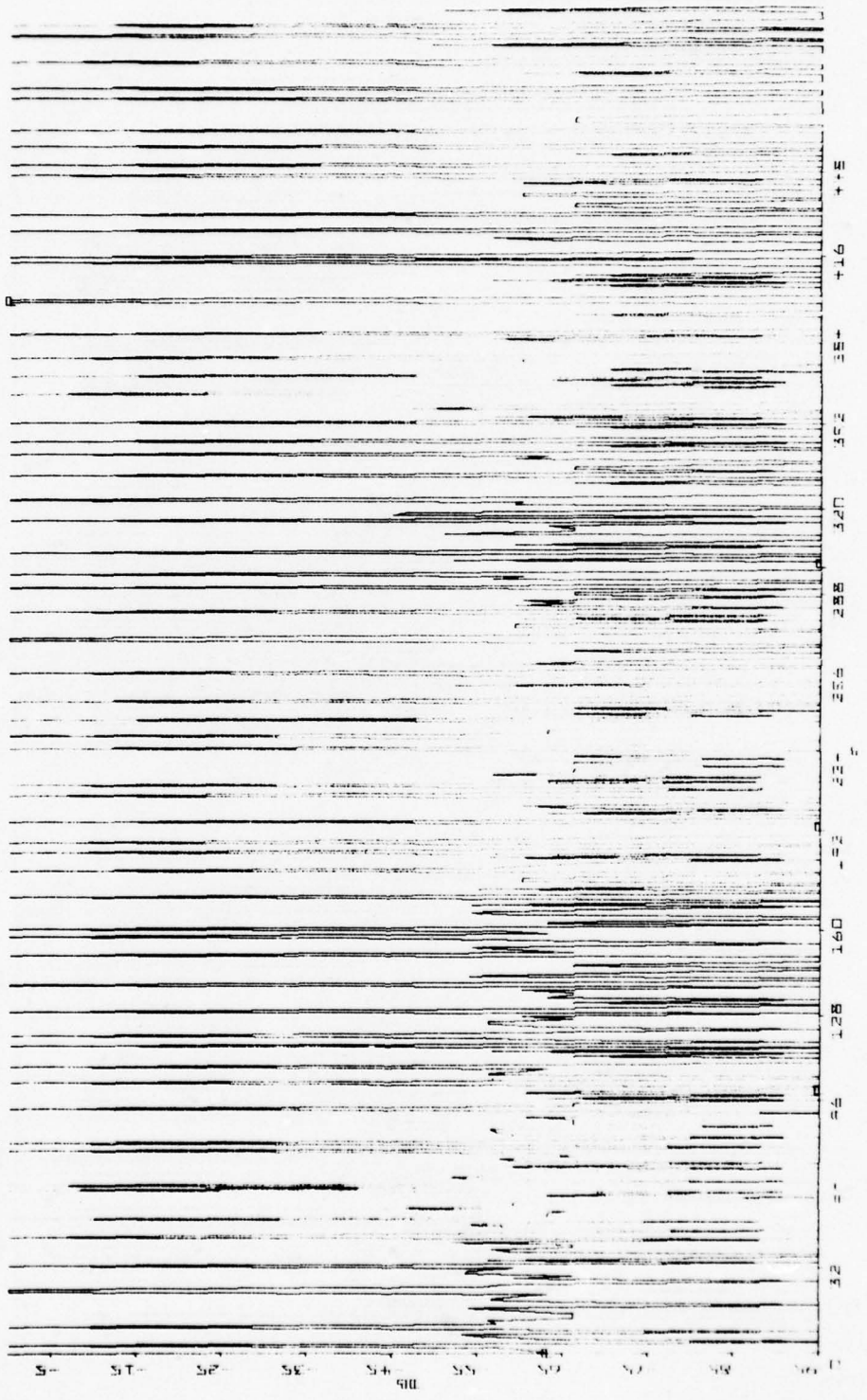


Figure A-9 FDM SPECTRUM (N=512, m=64, b=10)

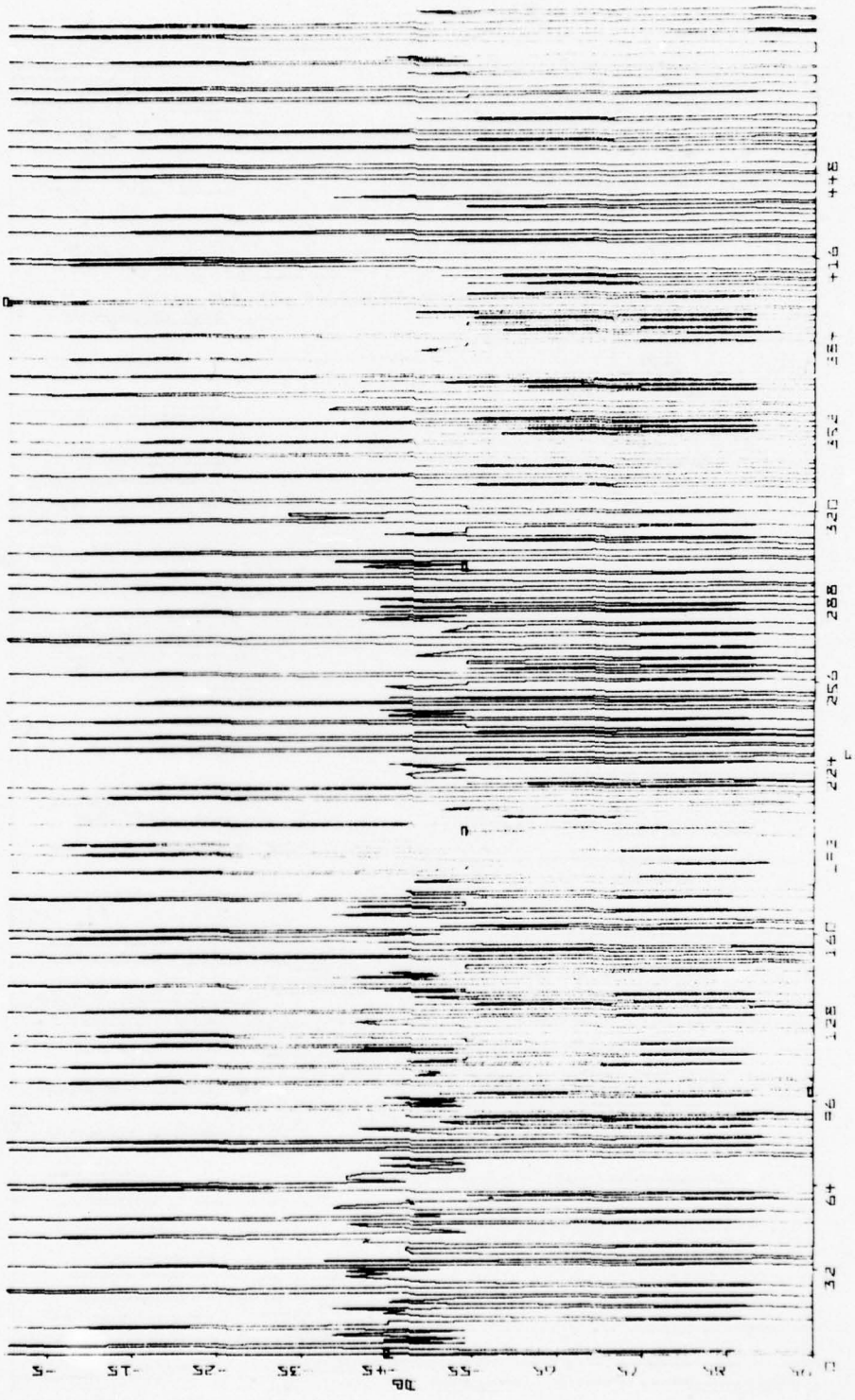


Figure A-10 FDM SPECTRUM (N=512, m=64, b=12)

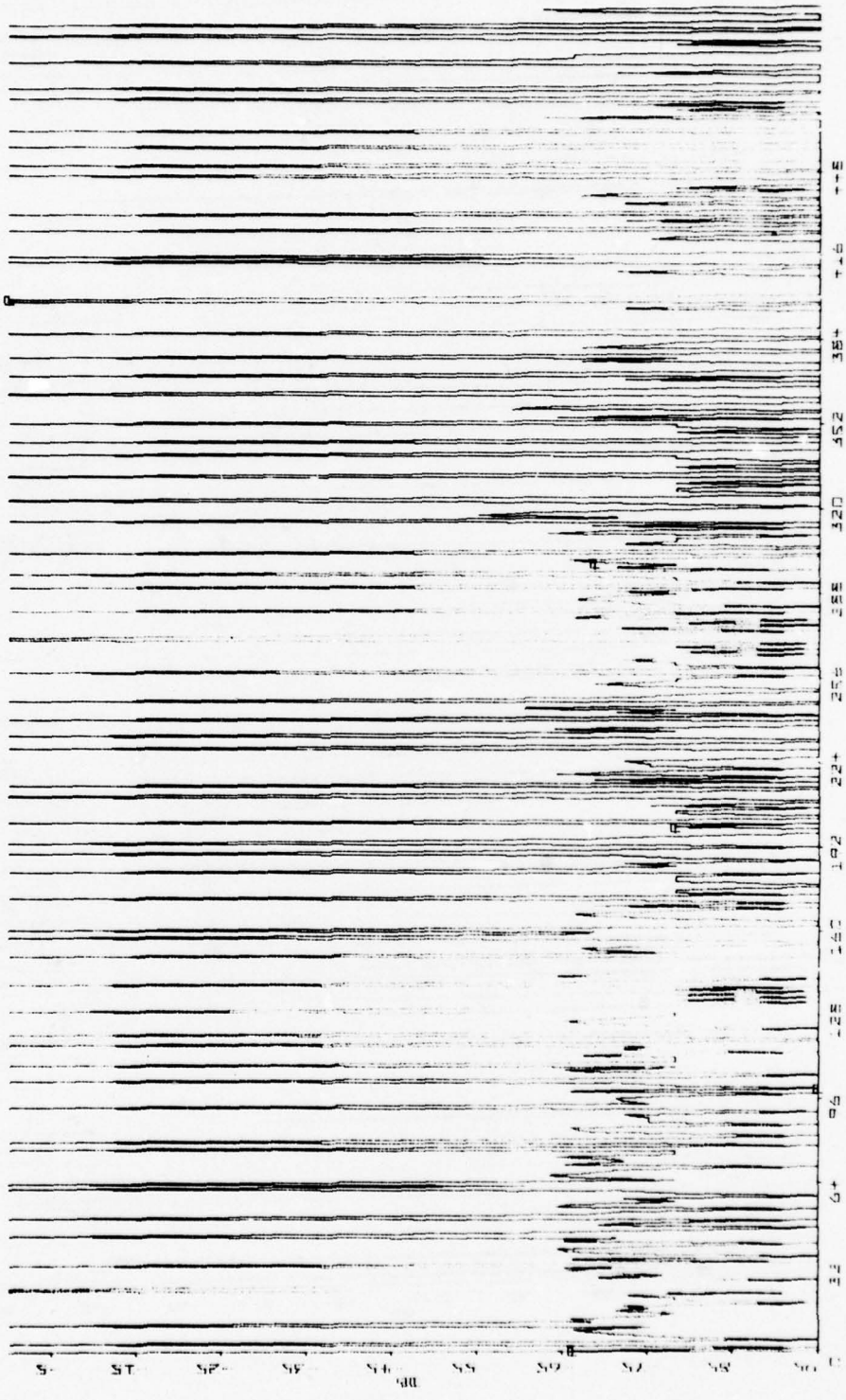


Figure A-11 FDM SPECTRUM (N=512, m=64, b=14)

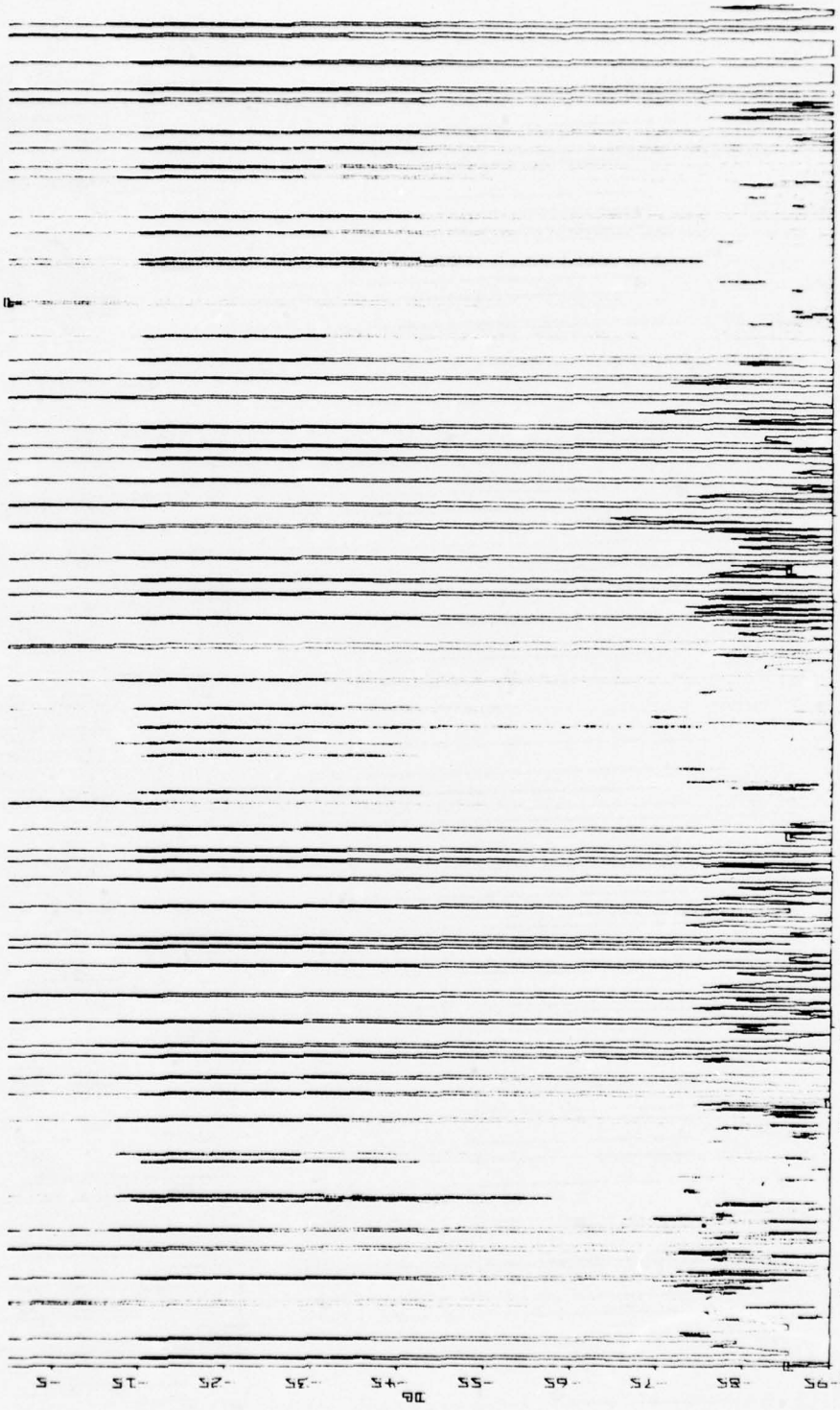


Figure A-12 FDM SPECTRUM (N=512, m=64, b=16)

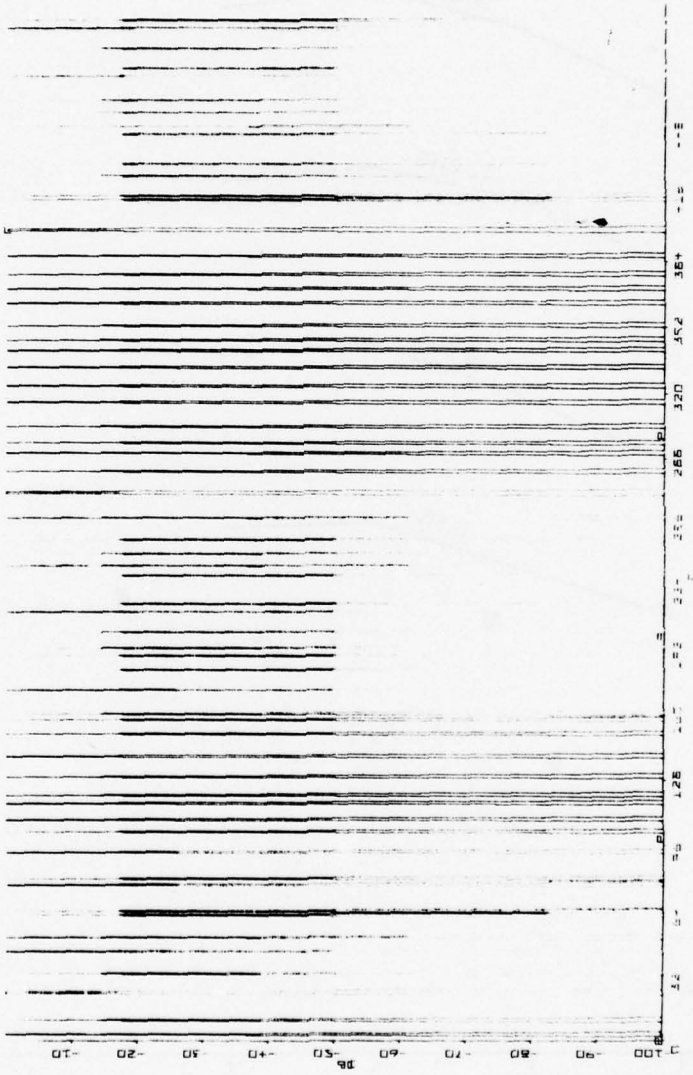


Figure A-13 FDM SPECTRUM (N=512, m=64, b=48)

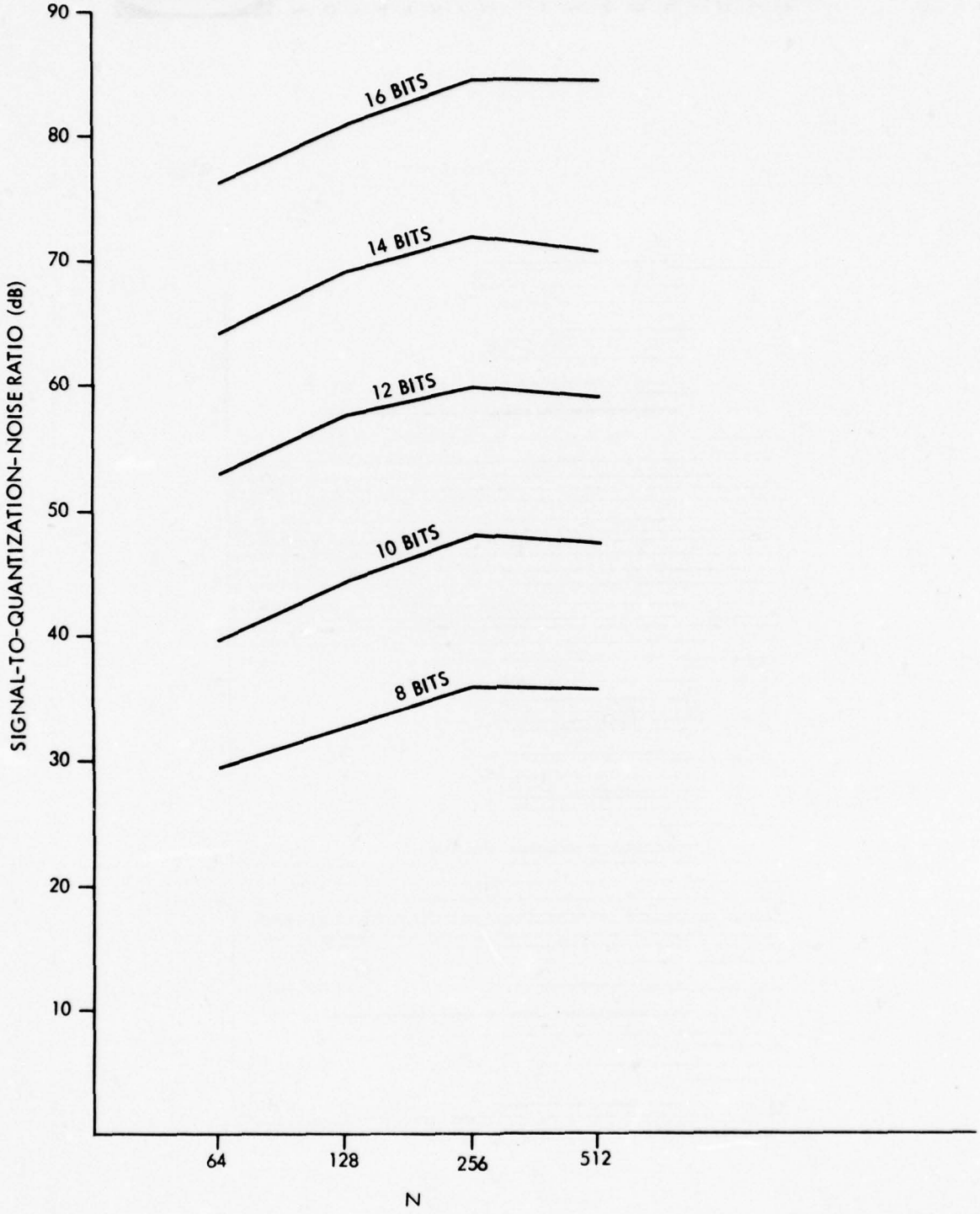


Figure A-14
SIGNAL-TO-NOISE RATIO VS. N (CONSTANT TONE AMPLITUDE)

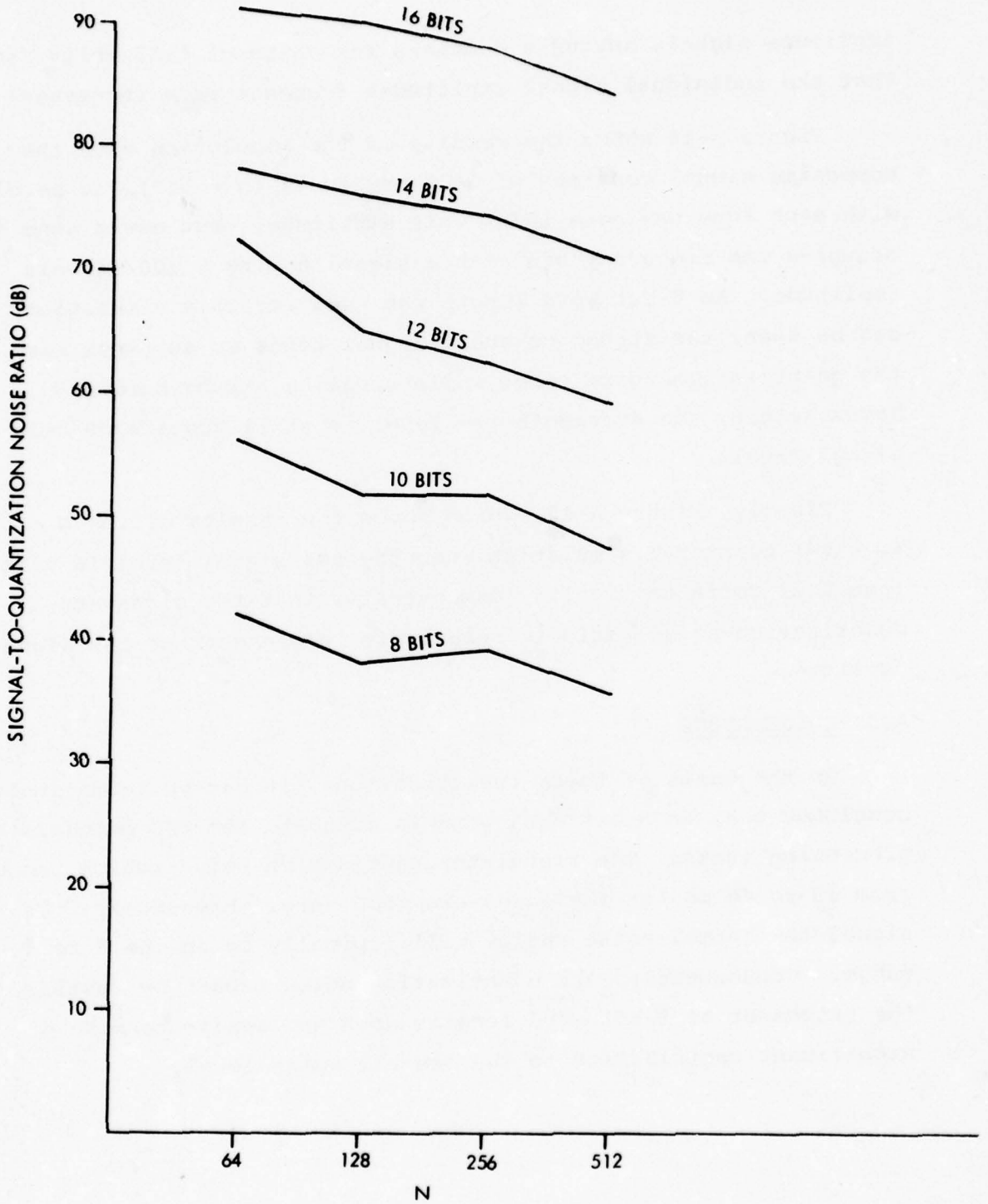


Figure A-15
SIGNAL-TO-NOISE RATIO VS. N COMPOSITE RMS (AMPLITUDE)

amplitude signals having a combined rms value of 1.25 volts (so that the individual signal amplitudes decrease as m increases).

Figure A-16 shows the results of the simulation when the composite signal consists of 64 8-ary tones (n = 512), as before, with each tone having a 10/64 volt amplitude, but now a tone jammer occupies one frequency bin with a signal having a 100/64 volt amplitude. An 8-bit word length was used for this simulation. As can be seen, the strong coherent signal tends to suppress some of the quantization noise terms while creating strong harmonics. Nevertheless, the average noise level is still about 33dB below the signal level.

Finally, Figure A-17, which shows the results of the N = 512 (m = 64) point FFT simulation when the rms signal level is varied from 0.25 volts to 7 volts, demonstrates that the signal-to-quantization-noise ratio is relatively independent of the signal amplitude.

A.3 CONCLUSIONS

On the basis of these investigations, it can be tentatively concluded that an 8-bit data word is adequate for SSS demodulator processing tasks. The signal-to-quantization noise ratios ranged from 33 to 40 dB for the cases examined here. Presumably, the signal-to-thermal-noise ratios will typically be in the 0 to 10 dB range. Consequently, the quantization noise caused by restricting the processor to 8-bit word lengths does not appear to make a significant contribution to the overall noise level.

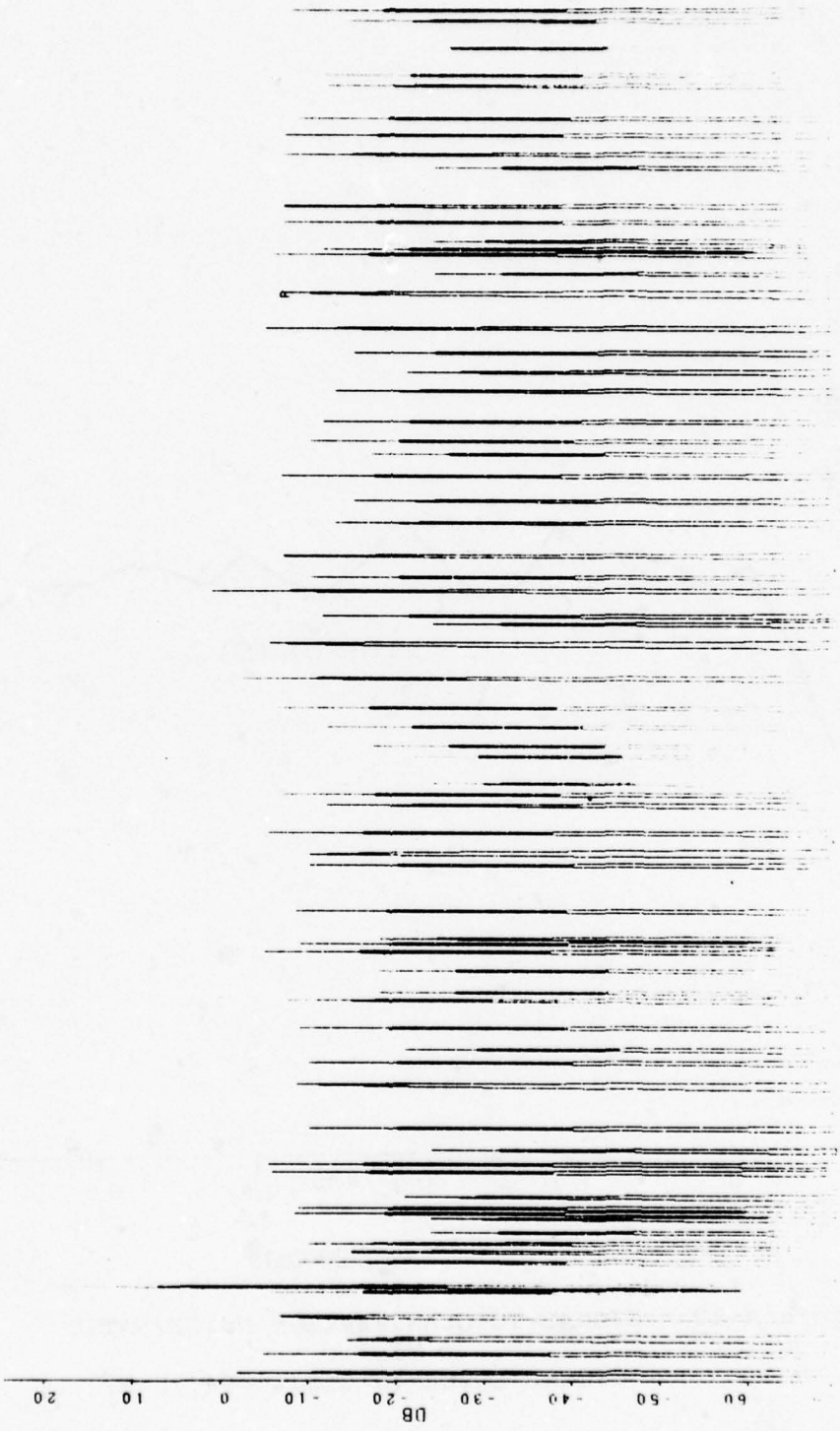


Figure A-16 FDM SIGNAL WITH STRONG (20 dB) JAMMER (N=512, m=64, b=8)

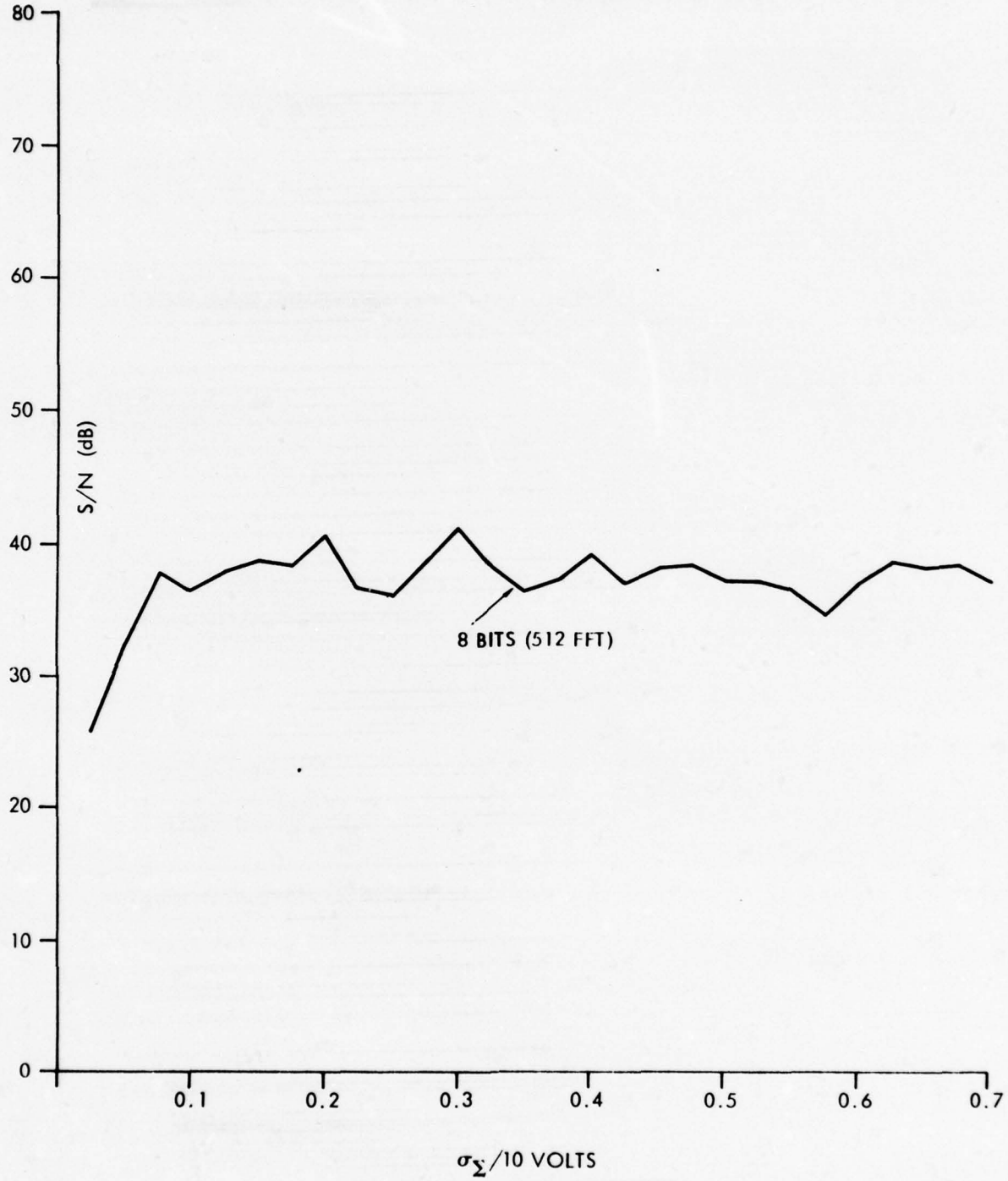


Figure A-17 SIGNAL-TO-QUANTIZATION NOISE RATIO VS. SIGNAL LEVEL (N=512, m=64, b=8)

APPENDIX B

CORE PROCESSOR CDL DESCRIPTION

B.1 INTRODUCTION

In order to facilitate both its evaluation and that of the micro-routines to be executed on it, the core processor was modeled using computer design language (CDL). Due to CDL limitations, some interim holding registers and additional clocking had to be added to the model. Nevertheless, the CDL model is a functionally exact model of the core processor. The core processor can execute an ALU operation in one clock cycle and an I/O or memory reference instruction in two cycles. The CDL model can effectively do the same but up to six subcycles may be used during a single master-clock cycle.

B.2 CDL ARCHITECTURE

The CDL model is broken down into two basic sections. The first is the DECLARATION section in which all the hardware for the processor is defined. The second section is the LABELED statement. These statements define the buses and system flow of the processor. It is in this second section that the clock labels are used. They indicate what information is to be on what bus at what time. In this particular model, six cycles are actually involved in an ALU function and one cycle is used to test if the proper number of iterations has been completed.

B.3 CDL MODEL DESCRIPTION

The first clock cycle, /P*START/, takes the contents specified by the control RAM address register, divides these 40 bits up into the proper CONTROL RAM fields, and then loads them into the proper registers.

After the first clock cycle is completed, two Major Labels are evaluated (/=RF14.EQ.7=/, /=RF14.LT.6=/). If the first condition is true, an ALU function is specified and the following procedure is executed:

The proper format for the information transfer from the ANS and WRYREG registers is specified during the second clock cycle /CLK(0)/. This information is stored in temporary holding registers denoted GPREG and WKREG.

In the third clock cycle, /CLK(1)/, a write clock is checked on both the working (WKREG) and general-purpose (GPREG) registers. If either is high, the proper register input is selected and the contents of GPREG or WKREG are stored into the specified location. The working and general-purpose register arrays are each modeled as an 8 x 8 memory array.

During the fourth cycle, /CLK(2)/, the ALU inputs are selected and stored in temporary holding registers denoted ALUA and ALUB.

The specified ALU function is then executed during the next cycle, /CLK(3)/, and the result stored in a nine-bit register denoted ANS.

The second major label, /=RF14.LT.6=/, indicates a memory or I/O operation. In this case, a further check is made to determine whether an I/O or memory reference is specified. In the case of an I/O operation, a further determination is made to see if it is an input or output operation and either the contents of the input register, INREG, are transferred to ANS or the contents of ANS are transferred to the output register, OUTREG. If a memory reference is specified, the store/fetch control RAM field is checked and the scratch-pad RAM address is defined, depending upon another RAM control field, either directly by the contents of one of the

working registers or by these contents incremented by 32 or 64. Once the proper address and store or fetch decisions have been made, the data are transferred.

At this point, both sections, the ALU function and the I/O or memory request, merge. The following instructions are common to both operations:

In the last CPU cycle, /CLK(4)/, the branch variable to be used in the next CPU cycle is checked and the next control RAM address is created.

There is one more label statement, /CLK(5)/, during which a check is made to determine if the proper number of iterations has been completed. This label has no equivalent in the hardware and simply is a software tool.

B.4 CDL SAMPLE RUN

The following pages illustrate a sample run of seven iterations incorporating both an I/O memory operation and an ALU operation.

Iteration	Starting Label	Function
1	/P*START/	Input operation (005_8 from INREG)
2	/P*START/	ALU function (ALUA.XOR.ALUB)
3	/P*START/	Memory store (RAM ADDRESS 7_8)
4	/P*START/	ALU function (ALUA.OR.ALUB)
5	/P*START/	Memory fetch (RAM ADDRESS 047_8)
6	/P*START/	ALU function (ALUA.ADD.ALUB)
7	/P*START/	Output operation (100_8 to OUTREG)

```
CALL,CDLP(INPUT=IDPTST7)


1
    1   $TRANSLATE
    2   C
    3   C
    4   C
    5   $MAIN
    6   C
    7   C
    8   C
    9   C
   10   C
   11   C
   12   C*******************************************************************************
   13   C
   14   C
   15   C
   16    CLOCK,P
   17   C
   18   C
   19   C
   20   C*******************************************************************************
   21   C
   22   C
   23   C
   24    REGISTER,
   25   1          RF1(2-0),          $ CONTROL RAM SEQUENCER BRANCH ADD
   26   1          RF2(7-0),          $ CONTROL RAM NEXT ADDRESS
   27   1          RF3(2-0),          $ WORKING REG WX OUTPUT SELECT
   28   1          RF4(2-0),          $ WORKING REG WY AND GENERAL PUR REG RA OUT
   29   1          RF5,               $ WORKING REGISTER WX/WY INPUT SELECT
   30   1          RF6(1-0),          $ 4:1 MUX INPUT FOR REGISTER ARRAY
   31   1          RF7,               $ WORKING REGISTER WRITE CLOCK
   32   1          RF8(2-0),          $ GENERAL PUR REG RB OUTPUT SELECT
   33   1          RF9,               $ GENERAL PUR REG RA/RB INPUT SELECT
   34   1          RF10,              $ GENERAL PUR REG WRITE CLOCK
   35   1          RF11(2-0),         $ ALU FUNCTION SELECT
   36   1          RF12(2-0),         $ ALU A,B MULTIPLEXER INPUT
   37   1          RF13,              $ CARRY INPUT INTO ALU
   38   1          RF14(2-0),         $ I/O BUFFER + MEMORY REQUEST
   39   1          RF15(1-0),         $ SPECIAL FUNCTION DECODE
   40   1          RF16               $ READ/WRITE CONTROL
   41    REGISTER,
   42   1          START,             $ START BIT FOR A RUN
   43   1          T(3-0),            $ SEQUENCER CLOCK
   44   1          RUNS(7-0),         $ COUNTS THE NUMBER OF RUNS
   45   1          ANS(10-0),         $ HOLDING REG FOR ALU + I/O,MEM REQUEST
   46   1          WRYREG(7-0),       $ HOLDING REG FOR WY OUTPUT
   47   1          GPRADD(2-0),       $ DUMMY REG NEEDED TO SPECIFY GP MEMORY
   48   1          WKRADD(2-0),       $ DUMMY REG NEEDED TO SPECIFY WK MEMORY
   49   1          GPREG(7-0),        $ HOLDING REG FOR INPUT TO GP MEMORY
   50   1          WKREG(7-0),        $ HOLDING REG FOR INPUT TO WK MEMORY
   51   1          ALUA(7-0),         $ A INPUT INTO ALU
   52   1          ALUB(7-0),         $ B INPUT INTO ALU
```

```
 53   1          BVREG(6-0),       $ BRANCH VARIABLE REGISTER
 54   1          INREG(7-0),       $ INPUT DATA REGISTER
 55   1          OUTREG(7-0),      $ OUTPUT DATA REGISTER
 56   1          ROMADD(7-0),      $ ADDRESS REG FOR CONTROL RAM
 57   1          RAMADD(7-0)       $ DUMMY REG NEEDED TO SPECIFY RAM MEM
 58   C
 59   C
 60   C
 61   C*************************************************************************************
 62   C
 63   C
 64   C
 65    DECODER,
 66   1          CLK=T
 67   C
 68   C
 69   C
 70   C*************************************************************************************
 71   C
 72   C
 73   C
 74    TERMINAL,
 75   1          ALUF(3-0)=RF13-RF11               $ DETERMINES ALU FUNCTION
 76   C
 77   C
 78   C
 79   C*************************************************************************************
 80   C
 81   C
 82   C
 83    MEMORY,
 84   1          RAM(RAMADD)=RAM(0-377,7-0),       $ SCRATCHPAD RAM 256 X 8
 85   1          ROM(ROMADD)=ROM(0-377,43-0),      $ CONTROL RAM 256 X 36
 86   1          ROM1(ROMADD)=ROM1(0-377,3-0),     $ CONTROL RAM EX 256 X 3
 87   1          GPRMEM(GPRADD)=GPRMEM(0-7,7-0),   $ GENERAL PUR MEM 8 X 8
 88   1          WKRMEM(WKRADD)=WKRMEM(0-7,7-0)    $ WORKING REG MEM 8 X 8
 89   C
 90   C
 91   C
 92   C*************************************************************************************
 93   C
 94   C          BRANCH CONDITION MULTIPLEXER
 95   C
 96    MULTIPLEXER,ROMBRA(1-0)=RF1;
 97   1          0,   RF2(7)-RF2(6);
 98   1          1,   BVREG(0)-BVREG(1);
 99   1          2,   BVREG(1)-BVREG(6);
100   1          3,   BVREG(2)-RF2(6);
101   1          4,   BVREG(3)-RF2(6);
102   1          5,   BVREG(4)-RF2(6);
103   1          6,   BVREG(5)-RF2(6);
104   1          7,   BVREG(6)-RF2(6)
105   C
106   C
107   C
108   C*************************************************************************************
109   C
110   C          4:1 GENERAL PURPOSE MULTIPLEXER INPUT SELECT
111   C
112    MULTIPLEXER,GPRMUX(7-0)=RF6;
113   1          0,   WRYREG;
114   1          1,   ANS;
115   1          2,   ANS.SHFTR.1;
116   1          3,   ANS.SHFTL.1
117   C
118   C
```

```
C
C*************************************************************************************
C
C          4:1 WORKING REGISTER MULTIPLEXER INPUT SELECT
C
 MULTIPLEXER,WKMUX(7-0)=RF6)
1           0,   ANS)
1           1,   WRYREG)
1           2,   WRYREG.SHFTR.1)
1           3,   WRYREG.SHFTL.1
C
C
C
C*************************************************************************************
C
C          GENERAL PURPOSE REGISTER RA/RB INPUT SELECT
C
 MULTIPLEXER,MUX1(2-0)=RF9)
1           0,   RF4)
1           1,   RF8
C
C
C
C*************************************************************************************
C
C          WORKING REGISTER WX/WY INPUT SELECT
C
 MULTIPLEXER,MUX2(2-0)=RF5)
1           0,   RF3)
1           1,   RF4
C
C
C
C*************************************************************************************
C
C          ALU A MULTIPLEXER INPUT
C
 MULTIPLEXER,ALUAMX(7-0)=RF12)
1           0,   GPRMEM(RF8))
1           1,   GPRMEM(RF4))
1           2,   GPRMEM(RF8))
1           3,   GPRMEM(RF4))
1           4,   GPRMEM(RF8))
1           5,   WKRMEM(RF3))
1           6,   GPRMEM(RF8))
1           7,   WKRMEM(RF3)
C
C
C
C*************************************************************************************
C
C          ALU B MULTIPLEXER INPUT
C
 MULTIPLEXER,ALUBMX(7-0)=RF12)
1           0,   WKRMEM(RF3))
1           1,   WKRMEM(RF3))
1           2,   WKRMEM(RF3).SHFTL.1)
1           3,   000)
1           4,   GPRMEM(RF4))
1           5,   GPRMEM(RF4))
1           6,   000)
1           7,   000
C
C
C
C*************************************************************************************
```

```
C
C           ALU FUNCTION SELECT
C
 MULTIPLEXER,SUMREG(10-0)=ALUF;
1            0,   ALUA.AND.ALUB;
1            1,   ALUA.OR.ALUB;
1            2,   ALUA'.AND.ALUB;
1            3,   ALUA.XOR.ALUB;
1            4,   ALUA.ADD.ALUB;
1            5,   (ALUA.SUB.ALUB).SUB.1;
1            6,   (ALUB.SUB.ALUA).SUB.1;
1            7,   ALUB';
1           10,   ALUA.AND.ALUB;
1           11,   ALUA.OR.ALUB;
1           12,   ALUA'.AND.ALUB;
1           13,   ALUA.XOR.ALUB;
1           14,   (ALUA.ADD.ALUB).COUNT.;
1           15,   ALUA.SUB.ALUB;
1           16,   ALUB.SUB.ALUA;
1           17,   ALUB'.COUNT.
C
C
C
C*************************************************************************************
C
C           RAM ADDRESS MODIFIER
C
 MULTIPLEXER,MUX3(7-0)=RF14;
1            0,   WKRMEM(RF4);
1            1,   WKRMEM(RF4).ADD.40;
1            2,   WKRMEM(RF4).ADD.100;
1            3,   WKRMEM(RF4);
1            4,   WKRMEM(RF4);
1            5,   WKRMEM(RF4);
1            6,   WKRMEM(RF4);
1            7,   WKRMEM(RF4)
C
C
C
C*************************************************************************************
C
C
C
 TERMINAL,
1            ADDRA1(7-0)=MUX3,      $ SELECTS PROPER ADDRESS FOR STORE FETCH
1            ADDRS1(2-0)=MUX1,      $ GENERAL PURPOSE REGISTER ADDRESS
1            ADDRS2(2-0)=MUX2       $ WORKING REGISTER ADDRESS
C
C
C
C*************************************************************************************
C*************************************************************************************
C
C           TAKES OUTPUT FROM ROM AND LOADS IT INTO REGISTERS
C
 /P$START/      RF1=ROM(ROMADD)(2-0),
                RF2=ROM(ROMADD)(12-3),
                RF3=ROM(ROMADD)(15-13),
                RF4=ROM(ROMADD)(20-16),
                RF5=ROM(ROMADD)(21),
                RF6=ROM(ROMADD)(23-22),
                RF7=ROM(ROMADD)(24),
                RF8=ROM(ROMADD)(27-25),
                RF9=ROM(ROMADD)(30),
                RF10=ROM(ROMADD)(31),
                RF11=ROM(ROMADD)(34-32),
```

```
                RF12=ROM(ROMADD)(37-35),
                RF13=ROM(ROMADD)(40),
                RF14=ROM(ROMADD)(43-41),
                RF15=ROM1(ROMADD)(1-0),
                RF16=ROM1(ROMADD)(2),
                START=0,T=0
C
C
C
C*************************************************************************************
C
C          IS THIS AN ALU FUNCTION -YES DO CLK(0)-CLK(3)
C
 /=RF14.EQ.7=/
C
C          EXCECUTE  4:1 INPUT MUX'S FOR REGISTER ARRAYS
C
    /CLK(0)/      GPREG=GPRMUX,START=0,T=T.COUNT.,WKREG=WKMUX
C
C          SELECT PROPER INPUT REGISTERS
C
    /CLK(1)/      IF(RF10.EQ.1)THEN(GPRMEM(ADDRS1)=GPREG),
                  IF(RF7.EQ.1)THEN(WKRMEM(ADDRS2)=WKREG),
                  T=T.COUNT.,ANS=0
C
C          SELECT PROPER INPUTS TO ALU
C
    /CLK(2)/      ALUA=ALUAMX,ALUB=ALURMX,START=0,T=T.COUNT.,
                  WRYREG=WKRMEM(RF4)
C
C          EXCECUTE ALU FUNCTION
C
    /CLK(3)/      ANS=SUMREG,T=T.COUNT.,RF14=6
 /=/
C
C
C
C*************************************************************************************
C
C          IS THIS A MEMORY OR I/O OPERATION-YES DO CLK(0)
C
 /=RF14.LT.6=/
C
C          EXCECUTE MEMORY OR I/O OPERATION
C
    /CLK(0)/      IF(RF14.LE.2)THEN(IF(RF16.EQ.0)THEN
                   (RAM(ADDRA1)=ANS)ELSE(ANS=RAM(ADDRA1))),
                  IF(RF14.EQ.4)THEN(ANS=INREG)
                   ELSE(IF(RF14.EQ.5)THEN(OUTREG=ANS)),
                  T=T.ADD.4,RF14=6,WRYREG=WKRMEM(RF4),
                  RAMADD=MUX3
 /=/
C
C
C
C*************************************************************************************
C
C          THIS SECTION CHECKS BRANCH VARIABLES
C          TO BE USED IN THE NEXT CPU CYCLE
C
 /CLK(4)/       IF(ANS(7).EQ.1)THEN(BVREG(0)=1)ELSE(BVREG(0)=0),
                IF(ANS(6).EQ.1)THEN(BVREG(1)=1)ELSE(BVREG(1)=0),
                IF(ANS(0).EQ.1)THEN(BVREG(2)=1)ELSE(BVREG(2)=0),
                IF(ANS(1).EQ.1)THEN(BVREG(3)=1)ELSE(BVREG(3)=0),
                IF(RF15.EQ.3)THEN(BVREG(4)=ANS(10)),
                IF(ANS.AND.777.EQ.000)THEN(BVREG(5)=1)ELSE(BVREG(5)=0),
```

```
317                     IF(ANS(5).EQ.1)THEN(BVREG(6)=1)ELSE(BVREG(6)=0),
318                     T=T.COUNT.,RUNS=RUNS.COUNT.,
319                     ROMADD=ROMBRA-RF2(5-0)  & NEXT CONTROL RAM ADD
320     C
321     C
322     C
323     C*****************************************************************************
324     C
325     C           CHECKS FOR THE PROPER NUMBER OF ITERATIONS
326     C
327      /CLK(5)/       IF(RUNS.EQ.7)THEN(START=0)ELSE(START=1),T=T.COUNT.
328     C
329     C
330     C
331     C*****************************************************************************
332     C
333     C
334     C
335      END
1
336     $SIMULATE
                                S I M U L A T I O N

PROGRAM LENGTH FOR SIMULATION IS 00114212
CODE SIZE ALLOCATION IS 00002345
MACHINE HARDWARE ALLOCATION  IN CDC WORDS IS 00000534
337     *OUTPUT       START,RUNS,ANS,T,
338                   RF1,RF2,RF3,RF4,
339                   RF5,RF6,RF7,RF8,
340                   RF9,RF10,RF11,RF12,
341                   RF13,RF14,RF15,RF16,
342                   GPRMEM(0),GPRMEM(1),GPRMEM(2),GPRMEM(3),
343                   GPRMEM(4),GPRMEM(5),GPRMEM(6),GPRMEM(7),
344                   WKRMEM(0),WKRMEM(1),WKRMEM(2),WKRMEM(3),
345                   WKRMEM(4),WKRMEM(5),WKRMEM(6),WKRMEM(7),
346                   GPREG,WKREG,ALUA,ALUB,
347                   WRYREG,BVREG,OUTREG,INREG,
348                   ROMADD,ROM(ROMADD),RAMADD,RAM(RAMADD)
349     *LOAD       START=1,
350                 GPRADD=00,
351                 WKRADD=00,
352                 GPREG=000,
353                 WKREG=000,
354                 GPRMEM(0-7)=0,0,0,0,0,0,0,0,
355                 WKRMEM(0-7)=0,0,0,0,0,0,7,0,
356                 ALUA=0,
357                 ALUB=0,
358                 T=10,
359                 ANS=0,
360                 ROMADD=000,
361                 RUNS=0,
362                 BVREG=000,
363                 WRYREG=0,
364                 RAM(47)=071,
365                 INREG=5,
366                 OUTREG=0,
367                 RAMADD=0
368     *LOAD       ROM(0)=400255264010,
369                 ROM(1)=701655264020,
370                 ROM(2)=000051264030,
371                 ROM(3)=700655264040,
372                 ROM(4)=100051264050,
373                 ROM(5)=700255264060,
374                 ROM(6)=500051264070,
375                 ROM1(0)=3,ROM1(1)=3,ROM1(2)=3,ROM1(3)=3,
```

```
  376              ROM1(4)=7,ROM1(5)=3,ROM1(6)=3
  377    $SIM    110,10
)SIMULATION ENDED WITH NO TRUE LABELS AND NO PENDING INTERRUPTS
  378    $XREFS    MAP=2
1  CDL CROSS REFERENCE MAP  PAGE     1
```

NAME	TYPE	STARTING BIT POS.	LINE DECLRD	SIZE	LSB OR 1ST WRD	MSB OR LAST WRD	LENGTH OR CLK PHASES	
ADDRA1	TERMINAL	00001050	229	8	0	7		
								REFS 297 297
ADDRS1	TERMINAL	00001057	230	3	0	2		
								REFS 272
ADDRS2	TERMINAL	00001066	231	3	0	2		
								REFS 273
ALUA	REGISTER	00000134	51	8	0	7		
								DEFS 278 356
								REFS 189 190 191 192 193 194 195 197
								200 201 202 203 346
ALUAMX	MUX	00000304	156	8	0	7		
								REFS 278
ALUB	REGISTER	00000144	52	8	0	7		
								DEFS 278 357
								REFS 189 190 191 192 193 194 195 196
								199 200 201 202 203 204 346
ALUBMX	MUX	00000405	172	8	0	7		
								REFS 278
ALUF	TERMINAL	00000001	75	4	0	3		
								REFS 188 188
ANS	REGISTER	00000065	45	9	0	8		
								DEFS 274 283 297 298 359
								REFS 114 115 116 125 297 299 311 312
								315 316 317 337
BVREG	REGISTER	00000154	53	7	0	6		
								DEFS 311 311 312 312 313 313 314 314
								316 317 317 362
								REFS 98 98 99 99 100 101 102 103
CLK	DECODER	00000051	66	4				
								REFS 268 268 272 272 278 278 283 283
								311 311 327 327
GPRADD	REGISTER	00000106	47	3	0	2		
								DEFS 350
								REFS 87
GPREG	REGISTER	00000114	49	8	0	7		
								DEFS 268 352
								REFS 272 346
GPRMEM	MEMORY	00030223	87	8	0	7	8	
								DEFS 272 354
								REFS 157 158 159 160 161 163 177 178
								342 342 343 343 343 343
GPRMUX	MUX	00000124	112	8	0	7		
								REFS 268
INREG	REGISTER	00000163	54	8	0	7		
								DEFS 365
								REFS 298 347
MUX1	MUX	00000236	136	3	0	2		
								REFS 230
MUX2	MUX	00000261	146	3	0	2		
								REFS 231
MUX3	MUX	00000737	212	8	0	7		
								REFS 229 301
OUTREG	REGISTER	00000173	55	8	0	7		
								DEFS 299 366
								REFS 347

NAME	TYPE	STARTING BIT POS.	LINE DECLRD	SIZE	LSB OR 1ST WRD	MSB OR LAST WRD	LENGTH OR CLK PHASES	
P	CLOCK		16				1	REFS 240
RAM	MEMORY	00000223	84	8	0	255	256	DEFS 297 364 REFS 297 348
RAMADD	REGISTER	00000213	57	8	0	7		DEFS 301 367 REFS 84 348 348
RF1	REGISTER	00000001	25	3	0	2		DEFS 240 REFS 96 96 338
RF10	REGISTER	00000032	34	1	0	0		DEFS 249 REFS 272 340
RF11	REGISTER	00000033	35	3	0	2		DEFS 250 REFS 75 340
RF12	REGISTER	00000036	36	3	0	2		DEFS 251 REFS 156 156 172 172 340
RF13	REGISTER	00000041	37	1	0	0		DEFS 252

1 CDL CROSS REFERENCE MAP PAGE 2

NAME	TYPE	STARTING BIT POS.	LINE DECLRD	SIZE	LSB OR 1ST WRD	MSB OR LAST WRD	LENGTH OR CLK PHASES	
RF14	REGISTER	00000042	38	3	0	2		REFS 75 341
RF15	REGISTER	00000045	39	2	0	1		DEFS 253 283 300 REFS 212 212 264 292 296 298 299 341
RF16	REGISTER	00000047	40	1	0	0		DEFS 254 REFS 315 341
RF2	REGISTER	00000004	26	8	0	7		DEFS 255 REFS 296 341
RF3	REGISTER	00000014	27	3	0	2		DEFS 241 REFS 97 97 100 101 102 103 104 319 33(
RF4	REGISTER	00000017	28	3	0	2		DEFS 242 REFS 147 162 164 173 174 175 338
RF5	REGISTER	00000022	29	1	0	0		DEFS 243 REFS 137 148 158 160 177 178 213 214 21 217 218 219 220 279 300 338
RF6	REGISTER	00000023	30	2	0	1		DEFS 244 REFS 146 146 339
RF7	REGISTER	00000025	31	1	0	0		DEFS 245 REFS 112 112 124 124 339
RF8	REGISTER	00000026	32	3	0	2		DEFS 246 REFS 273 339
RF9	REGISTER	00000031	33	1	0	0		DEFS 247 REFS 138 157 159 161 163 339
ROM	MEMORY	00004223	85	36	0	255	256	DEFS 248 REFS 136 136 340
								DEFS 368 369 370 371 372 373 374 REFS 240 241 242 243 244 245 246 247 2

ROMADD	REGISTER	00000203	56	8	0	7		
ROMBRA	MUX	00000013	96	2	0	1		
ROM1	MEMORY	00026223	86	4	0	255	256	
RUNS	REGISTER	00000055	44	8	0	7		
START	REGISTER	00000050	42	1	0	0		
SUMREG	MUX	00000504	188	9	0	8		
T	REGISTER	00000051	43	4	0	3		
WKMUX	MUX	00000171	124	8	0	7		
WKRADD	REGISTER	00000111	48	3	0	2		
WKREG	REGISTER	00000124	50	8	0	7		
WKRMEM	MEMORY	00030323	88	8	0	7	8	

1 CDL CROSS REFERENCE MAP PAGE 3

NAME	TYPE	STARTING BIT POS.	LINE SIZE DECLRD		LSB OR 1ST WRD	MSB OR LAST WRD	LENGTH OR CLK PHASES	
WRYREG	REGISTER	00000076	46	8	0	7		

1 TABLES

0SUBPROGRAM TABLE 1

	LABEL TABLE 1ST	LABEL TABLE LAST	SW LABEL TABLE 1ST	SW LABEL TABLE LAST	INT LABEL TABLE 1ST	INT LABEL TABLE LAST	START ADDR
MAIN	1	10	0	0	0	0	000000

0MAJOR LABEL TABLE 2

RANGE OF NORMAL LABELS

3	6
8	8

0LABEL TABLE 10

		LINE	
1.	/P$START/	240	001075
2.	/=RF14.EQ.7=/	264	001404
3.	/CLK(0)/	268	001415
4.	/CLK(1)/	272	001460
5.	/CLK(2)/	278	001547
6.	/CLK(3)/	283	001621
7.	/=RF14.LT.6=/	292	001657
8.	/CLK(0)/	296	001671
9.	/CLK(4)/	311	002033
10.	/CLK(5)/	327	002277

1

```
  379   $REPORT
  380   $FORMAT     LABEL(1)=START,RUNS,ANS,T,
```

	250	251	252	253	348				
DEFS	319	360							
REFS	85	86	240	241	242	243	244	245	246
	248	249	250	251	252	253	254	255	348
REFS	319								
DEFS	375	375	375	375	376	376	376		
REFS	254	255							
DEFS	318	361							
REFS	318	327	337						
DEFS	256	268	278	327	327	349			
REFS	240	337							
REFS	283								
DEFS	256	268	274	278	283	300	318	327	358
REFS	66	268	274	278	283	300	318	327	337
REFS	268								
DEFS	351								
REFS	88								
DEFS	268	353							
REFS	273	346							
DEFS	273	355							
REFS	162	164	173	174	175	213	214	215	216
	218	219	220	279	300	344	344	344	344
	345	345	345						
DEFS	279	300	363						
REFS	113	126	127	128	347				

```
381   1          RF1,RF2,RF3,RF4,
382   1          RF5,RF6,RF7,RF8,
383   1          RF9,RF10,RF11,RF12,
384   1          RF13,RF14,RF15,RF16,
385   1          GPRMEM(0),GPRMEM(1),GPRMEM(2),GPRMEM(3),
386   1          GPRMEM(4),GPRMEM(5),GPRMEM(6),GPRMEM(7),
387   1          WKRMEM(0),WKRMEM(1),WKRMEM(2),WKRMEM(3),
388   1          WKRMEM(4),WKRMEM(5),WKRMEM(6),WKRMEM(7),
389   1          GPREG,WKREG,ALUA,ALUB,
390   1          WRYREG,BVREG,OUTREG,INREG,
391   1          ROMADD,ROM(ROMADD),RAMADD,RAM(RAMADD)
392   $RUN
1
*********************************

      CLOCK TIME=     1          LABEL CYCLE=     1
                                      TRUE LABELS

   /P$START/
          START=0                  RUNS=000                 ANS=000                   T=00
            RF1=0                   RF2=001                 RF3=5                   RF4=5
            RF5=0                   RF6=1                   RF7=1                   RF8=5
            RF9=0                  RF10=1                  RF11=0                  RF12=0
           RF13=0                  RF14=4                  RF15=3                  RF16=0
  GPRMEM(000000)=000       GPRMEM(000001)=000       GPRMEM(000002)=000       GPRMEM(000003)=000
  GPRMEM(000004)=000       GPRMEM(000005)=000       GPRMEM(000006)=000       GPRMEM(000007)=000
  WKRMEM(000000)=000       WKRMEM(000001)=000       WKRMEM(000002)=000       WKRMEM(000003)=000
  WKRMEM(000004)=000       WKRMEM(000005)=007       WKRMEM(000006)=000       WKRMEM(000007)=000
         GPREG=000                WKREG=000                ALUA=000                ALUB=000
        WRYREG=000               BVREG=000               OUTREG=000               INREG=005
        ROMADD=000     ROM(ROMADD)=400255264010          RAMADD=000     RAM(RAMADD)=000

*********************************

      CLOCK TIME=     2          LABEL CYCLE=     2
                                      TRUE LABELS

   /CLK(0)/
          START=0                  RUNS=000                 ANS=005                   T=04
            RF1=0                   RF2=001                 RF3=5                   RF4=5
            RF5=0                   RF6=1                   RF7=1                   RF8=5
            RF9=0                  RF10=1                  RF11=0                  RF12=0
           RF13=0                  RF14=6                  RF15=3                  RF16=0
  GPRMEM(000000)=000       GPRMEM(000001)=000       GPRMEM(000002)=000       GPRMEM(000003)=000
  GPRMEM(000004)=000       GPRMEM(000005)=000       GPRMEM(000006)=000       GPRMEM(000007)=000
  WKRMEM(000000)=000       WKRMEM(000001)=000       WKRMEM(000002)=000       WKRMEM(000003)=000
  WKRMEM(000004)=000       WKRMEM(000005)=007       WKRMEM(000006)=000       WKRMEM(000007)=000
         GPREG=000                WKREG=000                ALUA=000                ALUB=000
        WRYREG=007               BVREG=000               OUTREG=000               INREG=005
        ROMADD=000     ROM(ROMADD)=400255264010          RAMADD=007     RAM(RAMADD)=000

*********************************

      CLOCK TIME=     3          LABEL CYCLE=     3
                                      TRUE LABELS

   /CLK(4)/
          START=0                  RUNS=001                 ANS=005                   T=03
            RF1=0                   RF2=001                 RF3=5                   RF4=5
            RF5=0                   RF6=1                   RF7=1                   RF8=5
            RF9=0                  RF10=1                  RF11=0                  RF12=0
           RF13=0                  RF14=6                  RF15=3                  RF16=0
  GPRMEM(000000)=000       GPRMEM(000001)=000       GPRMEM(000002)=000       GPRMEM(000003)=000
  GPRMEM(000004)=000       GPRMEM(000005)=000       GPRMEM(000006)=000       GPRMEM(000007)=000
  WKRMEM(000000)=000       WKRMEM(000001)=000       WKRMEM(000002)=000       WKRMEM(000003)=000
  WKRMEM(000004)=000       WKRMEM(000005)=007       WKRMEM(000006)=000       WKRMEM(000007)=000
```

```
        GPREG=000              WKREG=000              ALUA=000              ALUB=000
       WRYREG=007              BVREG=004            OUTREG=000             INREG=005
       ROMADD=001    ROM(ROMADD)=701655264020      RAMADD=007    RAM(RAMADD)=000

!******************************

    CLOCK TIME=     4       LABEL CYCLE=     4
                                TRUE LABELS

  /CLK(5)/
         START=1                RUNS=001               ANS=005                 T=06
           RF1=0                 RF2=001                RF3=5                 RF4=5
           RF5=0                 RF6=1                  RF7=1                 RF8=5
           RF9=0                RF10=1                 RF11=0                RF12=0
          RF13=0                RF14=6                 RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=000    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=000              WKREG=000              ALUA=000              ALUB=000
       WRYREG=007              BVREG=004            OUTREG=000             INREG=005
       ROMADD=001    ROM(ROMADD)=701655264020      RAMADD=007    RAM(RAMADD)=000

!******************************

    CLOCK TIME=     5       LABEL CYCLE=     5
                                TRUE LABELS

  /P*START/
         START=0                RUNS=001               ANS=005                 T=00
           RF1=0                 RF2=002                RF3=5                 RF4=5
           RF5=0                 RF6=1                  RF7=1                 RF8=5
           RF9=0                RF10=1                 RF11=3                RF12=0
          RF13=0                RF14=7                 RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=000    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=000              WKREG=000              ALUA=000              ALUB=000
       WRYREG=007              BVREG=004            OUTREG=000             INREG=005
       ROMADD=001    ROM(ROMADD)=701655264020      RAMADD=007    RAM(RAMADD)=000

!******************************

    CLOCK TIME=     6       LABEL CYCLE=     6
                                TRUE LABELS

  /CLK(0)/
         START=0                RUNS=001               ANS=005                 T=01
           RF1=0                 RF2=002                RF3=5                 RF4=5
           RF5=0                 RF6=1                  RF7=1                 RF8=5
           RF9=0                RF10=1                 RF11=3                RF12=0
          RF13=0                RF14=7                 RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=000    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=005              WKREG=007              ALUA=000              ALUB=000
       WRYREG=007              BVREG=004            OUTREG=000             INREG=005
       ROMADD=001    ROM(ROMADD)=701655264020      RAMADD=007    RAM(RAMADD)=000

!******************************

    CLOCK TIME=     7       LABEL CYCLE=     7
                                TRUE LABELS
```

```
/CLK(1)/
        START=0          RUNS=001          ANS=000           T=02
          RF1=0           RF2=002          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=3          RF12=0
         RF13=0          RF14=7           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005          WKREG=007           ALUA=000           ALUB=000
        WRYREG=007          BVREG=004         OUTREG=000         INREG=005
        ROMADD=001  ROM(ROMADD)=701655264020  RAMADD=007  RAM(RAMADD)=000

**********

    CLOCK TIME=     8        LABEL CYCLE=     8
                                 TRUE LABELS

/CLK(2)/
        START=0          RUNS=001          ANS=000           T=03
          RF1=0           RF2=002          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=3          RF12=0
         RF13=0          RF14=7           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005          WKREG=007           ALUA=005           ALUB=007
        WRYREG=007          BVREG=004         OUTREG=000         INREG=005
        ROMADD=001  ROM(ROMADD)=701655264020  RAMADD=007  RAM(RAMADD)=000

**********

    CLOCK TIME=     9        LABEL CYCLE=     9
                                 TRUE LABELS

/CLK(3)/
        START=0          RUNS=001          ANS=002           T=04
          RF1=0           RF2=002          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=3          RF12=0
         RF13=0          RF14=6           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005          WKREG=007           ALUA=005           ALUB=007
        WRYREG=007          BVREG=004         OUTREG=000         INREG=005
        ROMADD=001  ROM(ROMADD)=701655264020  RAMADD=007  RAM(RAMADD)=000

**********

    CLOCK TIME=    10        LABEL CYCLE=    10
                                 TRUE LABELS

/CLK(4)/
        START=0          RUNS=002          ANS=002           T=05
          RF1=0           RF2=002          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=3          RF12=0
         RF13=0          RF14=6           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
```

```
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
        GPREG=005               WKREG=007               ALUA=005                ALUB=007
        WRYREG=007              BVREG=010             OUTREG=000              INREG=004
        ROMADD=002    ROM(ROMADD)=000051264030        RAMADD=007      RAM(RAMADD)=000

**********

    CLOCK TIME=    11       LABEL CYCLE=    11
                                TRUE LABELS

  /CLK(5)/
        START=1                 RUNS=002                ANS=002                 T=06
          RF1=0                 RF2=002                 RF3=5                 RF4=5
          RF5=0                 RF6=1                   RF7=1                 RF8=5
          RF9=0                 RF10=1                  RF11=3                RF12=0
         RF13=0                 RF14=6                  RF15=3                RF16=0
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=005      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
        GPREG=005               WKREG=007               ALUA=005                ALUB=007
        WRYREG=007              BVREG=010             OUTREG=000              INREG=005
        ROMADD=002    ROM(ROMADD)=000051264030        RAMADD=007      RAM(RAMADD)=000

**********

    CLOCK TIME=    12       LABEL CYCLE=    12
                                TRUE LABELS

  /P$START/
        START=0                 RUNS=002                ANS=002                 T=00
          RF1=0                 RF2=003                 RF3=5                 RF4=5
          RF5=0                 RF6=1                   RF7=0                 RF8=5
          RF9=0                 RF10=0                  RF11=0                RF12=0
         RF13=0                 RF14=0                  RF15=3                RF16=0
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=005      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
        GPREG=005               WKREG=007               ALUA=005                ALUB=007
        WRYREG=007              BVREG=010             OUTREG=000              INREG=005
        ROMADD=002    ROM(ROMADD)=000051264030        RAMADD=007      RAM(RAMADD)=000

**********

    CLOCK TIME=    13       LABEL CYCLE=    13
                                TRUE LABELS

  /CLK(0)/
        START=0                 RUNS=002                ANS=002                 T=04
          RF1=0                 RF2=003                 RF3=5                 RF4=5
          RF5=0                 RF6=1                   RF7=0                 RF8=5
          RF9=0                 RF10=0                  RF11=0                RF12=0
         RF13=0                 RF14=6                  RF15=3                RF16=0
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=005      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
        GPREG=005               WKREG=007               ALUA=005                ALUB=007
        WRYREG=007              BVREG=010             OUTREG=000              INREG=005
        ROMADD=002    ROM(ROMADD)=000051264030        RAMADD=007      RAM(RAMADD)=002

**********

    CLOCK TIME=    14       LABEL CYCLE=    14
                                TRUE LABELS
```

```
/CLK(4)/
        START=0          RUNS=003          ANS=002          T=05
          RF1=0           RF2=003          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=0           RF8=5
          RF9=0          RF10=0           RF11=0          RF12=0
         RF13=0          RF14=6           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005         WKREG=007         ALUA=005         ALUB=007
        WRYREG=007         BVREG=010       OUTREG=000       INREG=005
        ROMADD=003  ROM(ROMADD)=700655264040  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    15        LABEL CYCLE=    15
                                 TRUE LABELS

/CLK(5)/
        START=1          RUNS=003          ANS=002          T=06
          RF1=0           RF2=003          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=0           RF8=5
          RF9=0          RF10=0           RF11=0          RF12=0
         RF13=0          RF14=6           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005         WKREG=007         ALUA=005         ALUB=007
        WRYREG=007         BVREG=010       OUTREG=000       INREG=005
        ROMADD=003  ROM(ROMADD)=700655264040  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    16        LABEL CYCLE=    16
                                 TRUE LABELS

/P#START/
        START=0          RUNS=003          ANS=002          T=00
          RF1=0           RF2=004          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=1          RF12=0
         RF13=0          RF14=7           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=005         WKREG=007         ALUA=005         ALUB=007
        WRYREG=007         BVREG=010       OUTREG=000       INREG=005
        ROMADD=003  ROM(ROMADD)=700655264040  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    17        LABEL CYCLE=    17
                                 TRUE LABELS

/CLK(0)/
        START=0          RUNS=003          ANS=002          T=01
          RF1=0           RF2=004          RF3=5           RF4=5
          RF5=0           RF6=1            RF7=1           RF8=5
          RF9=0          RF10=1           RF11=1          RF12=0
         RF13=0          RF14=7           RF15=3          RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=005  GPRMEM(000006)=000  GPRMEM(000007)=000
```

```
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
       GPREG=002             WKREG=007             ALUA=005              ALUB=007
      WRYREG=007             BVREG=010          OUTREG=000            INREG=005
      ROMADD=003       ROM(ROMADD)=700655264040 RAMADD=007       RAM(RAMADD)=002

********************************

    CLOCK TIME=    18        LABEL CYCLE=    18
                                 TRUE LABELS

 /CLK(1)/
       START=0                RUNS=003              ANS=000                T=02
         RF1=0                 RF2=004               RF3=5                RF4=5
         RF5=0                 RF6=1                 RF7=1                RF8=5
         RF9=0                RF10=1                RF11=1               RF12=0
        RF13=0                RF14=7                RF15=3               RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=002    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
       GPREG=002             WKREG=007             ALUA=005              ALUB=007
      WRYREG=007             BVREG=010          OUTREG=000            INREG=005
      ROMADD=003       ROM(ROMADD)=700655264040 RAMADD=007       RAM(RAMADD)=002

********************************

    CLOCK TIME=    19        LABEL CYCLE=    19
                                 TRUE LABELS

 /CLK(2)/
       START=0                RUNS=003              ANS=000                T=03
         RF1=0                 RF2=004               RF3=5                RF4=5
         RF5=0                 RF6=1                 RF7=1                RF8=5
         RF9=0                RF10=1                RF11=1               RF12=0
        RF13=0                RF14=7                RF15=3               RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=002    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
       GPREG=002             WKREG=007             ALUA=002              ALUB=007
      WRYREG=007             BVREG=010          OUTREG=000            INREG=005
      ROMADD=003       ROM(ROMADD)=700655264040 RAMADD=007       RAM(RAMADD)=002

********************************

    CLOCK TIME=    20        LABEL CYCLE=    20
                                 TRUE LABELS

 /CLK(3)/
       START=0                RUNS=003              ANS=007                T=04
         RF1=0                 RF2=004               RF3=5                RF4=5
         RF5=0                 RF6=1                 RF7=1                RF8=5
         RF9=0                RF10=1                RF11=1               RF12=0
        RF13=0                RF14=6                RF15=3               RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=002    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
       GPREG=002             WKREG=007             ALUA=002              ALUB=007
      WRYREG=007             BVREG=010          OUTREG=000            INREG=005
      ROMADD=003       ROM(ROMADD)=700655264040 RAMADD=007       RAM(RAMADD)=002

********************************

    CLOCK TIME=    21        LABEL CYCLE=    21
                                 TRUE LABELS
```

```
/CLK(4)/
        START=0          RUNS=004          ANS=007          T=05
          RF1=0           RF2=004           RF3=5          RF4=5
          RF5=0           RF6=1             RF7=1          RF8=5
          RF9=0          RF10=1            RF11=1         RF12=0
         RF13=0          RF14=6            RF15=3         RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=002  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=002          WKREG=007           ALUA=002          ALUB=007
       WRYREG=007          BVREG=014        OUTREG=000         INREG=005
       ROMADD=004  ROM(ROMADD)=100051264050  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    22      LABEL CYCLE=    22
                                TRUE LABELS

/CLK(5)/
        START=1          RUNS=004          ANS=007          T=06
          RF1=0           RF2=004           RF3=5          RF4=5
          RF5=0           RF6=1             RF7=1          RF8=5
          RF9=0          RF10=1            RF11=1         RF12=0
         RF13=0          RF14=6            RF15=3         RF16=0
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=002  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=002          WKREG=007           ALUA=002          ALUB=007
       WRYREG=007          BVREG=014        OUTREG=000         INREG=005
       ROMADD=004  ROM(ROMADD)=100051264050  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    23      LABEL CYCLE=    23
                                TRUE LABELS

/P&START/
        START=0          RUNS=004          ANS=007          T=00
          RF1=0           RF2=005           RF3=5          RF4=5
          RF5=0           RF6=1             RF7=0          RF8=5
          RF9=0          RF10=0            RF11=0         RF12=0
         RF13=0          RF14=1            RF15=3         RF16=1
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
GPRMEM(000004)=000  GPRMEM(000005)=002  GPRMEM(000006)=000  GPRMEM(000007)=000
WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
         GPREG=002          WKREG=007           ALUA=002          ALUB=007
       WRYREG=007          BVREG=014        OUTREG=000         INREG=005
       ROMADD=004  ROM(ROMADD)=100051264050  RAMADD=007  RAM(RAMADD)=002

**********

    CLOCK TIME=    24      LABEL CYCLE=    24
                                TRUE LABELS

/CLK(0)/
        START=0          RUNS=004          ANS=071          T=04
          RF1=0           RF2=005           RF3=5          RF4=5
          RF5=0           RF6=1             RF7=0          RF8=5
          RF9=0          RF10=0            RF11=0         RF12=0
         RF13=0          RF14=6            RF15=3         RF16=1
GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
```

```
GPRMEM(000004)=000      GPRMEM(000005)=002      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
          GPREG=002               WKREG=007               ALUA=002                ALUB=007
         WRYREG=007               BVREG=014            OUTREG=000               INREG=005
         ROMADD=004   ROM(ROMADD)=100051264050        RAMADD=047     RAM(RAMADD)=071

*********************************

     CLOCK TIME=    25        LABEL CYCLE=    25
                                    TRUE LABELS

  /CLK(4)/
          START=0                 RUNS=005                ANS=071                T=05
            RF1=0                  RF2=005                RF3=5                RF4=5
            RF5=0                  RF6=1                  RF7=0                RF8=5
            RF9=0                 RF10=0                 RF11=0               RF12=0
           RF13=0                 RF14=6                 RF15=3               RF16=1
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=002      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
          GPREG=002               WKREG=007               ALUA=002                ALUB=007
         WRYREG=007               BVREG=104            OUTREG=000               INREG=005
         ROMADD=005   ROM(ROMADD)=702255264060        RAMADD=047     RAM(RAMADD)=071

*********************************

     CLOCK TIME=    26        LABEL CYCLE=    26
                                    TRUE LABELS

  /CLK(5)/
          START=1                 RUNS=005                ANS=071                T=06
            RF1=0                  RF2=005                RF3=5                RF4=5
            RF5=0                  RF6=1                  RF7=0                RF8=5
            RF9=0                 RF10=0                 RF11=0               RF12=0
           RF13=0                 RF14=6                 RF15=3               RF16=1
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=002      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
          GPREG=002               WKREG=007               ALUA=002                ALUB=007
         WRYREG=007               BVREG=104            OUTREG=000               INREG=005
         ROMADD=005   ROM(ROMADD)=702255264060        RAMADD=047     RAM(RAMADD)=071

*********************************

     CLOCK TIME=    27        LABEL CYCLE=    27
                                    TRUE LABELS

  /P*START/
          START=0                 RUNS=005                ANS=071                T=00
            RF1=0                  RF2=006                RF3=5                RF4=5
            RF5=0                  RF6=1                  RF7=1                RF8=5
            RF9=0                 RF10=1                 RF11=4               RF12=0
           RF13=0                 RF14=7                 RF15=3               RF16=0
GPRMEM(000000)=000      GPRMEM(000001)=000      GPRMEM(000002)=000      GPRMEM(000003)=000
GPRMEM(000004)=000      GPRMEM(000005)=002      GPRMEM(000006)=000      GPRMEM(000007)=000
WKRMEM(000000)=000      WKRMEM(000001)=000      WKRMEM(000002)=000      WKRMEM(000003)=000
WKRMEM(000004)=000      WKRMEM(000005)=007      WKRMEM(000006)=000      WKRMEM(000007)=000
          GPREG=002               WKREG=007               ALUA=002                ALUB=007
         WRYREG=007               BVREG=104            OUTREG=000               INREG=005
         ROMADD=005   ROM(ROMADD)=702255264060        RAMADD=047     RAM(RAMADD)=071

*********************************
```

```
     CLOCK TIME=    28          LABEL CYCLE=    28
                                    TRUE LABELS

  /CLK(0)/
          START=0                    RUNS=005                   ANS=071                    T=01
            RF1=0                     RF2=006                   RF3=5                    RF4=5
            RF5=0                     RF6=1                     RF7=1                    RF8=5
            RF9=0                    RF10=1                    RF11=4                   RF12=0
           RF13=0                    RF14=7                    RF15=3                   RF16=0
  GPRMEM(000000)=000         GPRMEM(000001)=000         GPRMEM(000002)=000         GPRMEM(000003)=000
  GPRMEM(000004)=000         GPRMEM(000005)=002         GPRMEM(000006)=000         GPRMEM(000007)=000
  WKRMEM(000000)=000         WKRMEM(000001)=000         WKRMEM(000002)=000         WKRMEM(000003)=000
  WKRMEM(000004)=000         WKRMEM(000005)=007         WKRMEM(000006)=000         WKRMEM(000007)=000
           GPREG=071                 WKREG=007                  ALUA=002                 ALUB=007
         WRYREG=007                  BVREG=104                 OUTREG=000                INREG=005
         ROMADD=005          ROM(ROMADD)=702255264060         RAMADD=047          RAM(RAMADD)=071

**********

     CLOCK TIME=    29          LABEL CYCLE=    29
                                    TRUE LABELS

  /CLK(1)/
          START=0                    RUNS=005                   ANS=000                    T=02
            RF1=0                     RF2=006                   RF3=5                    RF4=5
            RF5=0                     RF6=1                     RF7=1                    RF8=5
            RF9=0                    RF10=1                    RF11=4                   RF12=0
           RF13=0                    RF14=7                    RF15=3                   RF16=0
  GPRMEM(000000)=000         GPRMEM(000001)=000         GPRMEM(000002)=000         GPRMEM(000003)=000
  GPRMEM(000004)=000         GPRMEM(000005)=071         GPRMEM(000006)=000         GPRMEM(000007)=000
  WKRMEM(000000)=000         WKRMEM(000001)=000         WKRMEM(000002)=000         WKRMEM(000003)=000
  WKRMEM(000004)=000         WKRMEM(000005)=007         WKRMEM(000006)=000         WKRMEM(000007)=000
           GPREG=071                 WKREG=007                  ALUA=002                 ALUB=007
         WRYREG=007                  BVREG=104                 OUTREG=000                INREG=005
         ROMADD=005          ROM(ROMADD)=702255264060         RAMADD=047          RAM(RAMADD)=071

**********

     CLOCK TIME=    30          LABEL CYCLE=    30
                                    TRUE LABELS

  /CLK(2)/
          START=0                    RUNS=005                   ANS=000                    T=03
            RF1=0                     RF2=006                   RF3=5                    RF4=5
            RF5=0                     RF6=1                     RF7=1                    RF8=5
            RF9=0                    RF10=1                    RF11=4                   RF12=0
           RF13=0                    RF14=7                    RF15=3                   RF16=0
  GPRMEM(000000)=000         GPRMEM(000001)=000         GPRMEM(000002)=000         GPRMEM(000003)=000
  GPRMEM(000004)=000         GPRMEM(000005)=071         GPRMEM(000006)=000         GPRMEM(000007)=000
  WKRMEM(000000)=000         WKRMEM(000001)=000         WKRMEM(000002)=000         WKRMEM(000003)=000
  WKRMEM(000004)=000         WKRMEM(000005)=007         WKRMEM(000006)=000         WKRMEM(000007)=000
           GPREG=071                 WKREG=007                  ALUA=071                 ALUB=007
         WRYREG=007                  BVREG=104                 OUTREG=000                INREG=005
         ROMADD=005          ROM(ROMADD)=702255264060         RAMADD=047          RAM(RAMADD)=071

**********

     CLOCK TIME=    31          LABEL CYCLE=    31
                                    TRUE LABELS

  /CLK(3)/
          START=0                    RUNS=005                   ANS=100                    T=04
            RF1=0                     RF2=006                   RF3=5                    RF4=5
            RF5=0                     RF6=1                     RF7=1                    RF8=5
            RF9=0                    RF10=1                    RF11=4                   RF12=0
           RF13=0                    RF14=6                    RF15=3                   RF16=0
```

```
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=071    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=071             WKREG=007             ALUA=071              ALUB=007
      WRYREG=007             BVREG=104            OUTREG=000            INREG=005
      ROMADD=005    ROM(ROMADD)=702255264060      RAMADD=047    RAM(RAMADD)=071

**********

   CLOCK TIME=    32        LABEL CYCLE=    32
                                TRUE LABELS

 /CLK(4)/
      START=0                RUNS=006              ANS=100                T=05
        RF1=0                 RF2=006              RF3=5                 RF4=5
        RF5=0                 RF6=1                RF7=1                 RF8=5
        RF9=0                RF10=1                RF11=4                RF12=0
       RF13=0                RF14=6                RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=071    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=071             WKREG=007             ALUA=071              ALUB=007
      WRYREG=007             BVREG=002            OUTREG=000            INREG=005
      ROMADD=006    ROM(ROMADD)=500051264070      RAMADD=047    RAM(RAMADD)=071

**********

   CLOCK TIME=    33        LABEL CYCLE=    33
                                TRUE LABELS

 /CLK(5)/
      START=1                RUNS=006              ANS=100                T=06
        RF1=0                 RF2=006              RF3=5                 RF4=5
        RF5=0                 RF6=1                RF7=1                 RF8=5
        RF9=0                RF10=1                RF11=4                RF12=0
       RF13=0                RF14=6                RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=071    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=071             WKREG=007             ALUA=071              ALUB=007
      WRYREG=007             BVREG=002            OUTREG=000            INREG=005
      ROMADD=006    ROM(ROMADD)=500051264070      RAMADD=047    RAM(RAMADD)=071

**********

   CLOCK TIME=    34        LABEL CYCLE=    34
                                TRUE LABELS

 /P#START/
      START=0                RUNS=006              ANS=100                T=00
        RF1=0                 RF2=007              RF3=5                 RF4=5
        RF5=0                 RF6=1                RF7=0                 RF8=5
        RF9=0                RF10=0                RF11=0                RF12=0
       RF13=0                RF14=5                RF15=3                RF16=0
GPRMEM(000000)=000    GPRMEM(000001)=000    GPRMEM(000002)=000    GPRMEM(000003)=000
GPRMEM(000004)=000    GPRMEM(000005)=071    GPRMEM(000006)=000    GPRMEM(000007)=000
WKRMEM(000000)=000    WKRMEM(000001)=000    WKRMEM(000002)=000    WKRMEM(000003)=000
WKRMEM(000004)=000    WKRMEM(000005)=007    WKRMEM(000006)=000    WKRMEM(000007)=000
        GPREG=071             WKREG=007             ALUA=071              ALUB=007
      WRYREG=007             BVREG=002            OUTREG=000            INREG=005
      ROMADD=006    ROM(ROMADD)=500051264070      RAMADD=047    RAM(RAMADD)=071

**********
```

```
      CLOCK TIME=    35          LABEL CYCLE=    35
                                   TRUE LABELS

   /CLK(0)/
          START=0                  RUNS=006                 ANS=100                  T=04
            RF1=0                   RF2=007                 RF3=5                  RF4=5
            RF5=0                   RF6=1                   RF7=0                  RF8=5
            RF9=0                  RF10=0                  RF11=0                 RF12=0
           RF13=0                  RF14=6                  RF15=3                 RF16=0
 GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
 GPRMEM(000004)=000  GPRMEM(000005)=071  GPRMEM(000006)=000  GPRMEM(000007)=000
 WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
 WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
          GPREG=071              WKREG=007               ALUA=071               ALUB=007
         WRYREG=007              BVREG=002              OUTREG=100              INREG=005
         ROMADD=006  ROM(ROMADD)=500051264070           RAMADD=007  RAM(RAMADD)=002

**********************************

      CLOCK TIME=    36          LABEL CYCLE=    36
                                   TRUE LABELS

   /CLK(4)/
          START=0                  RUNS=007                 ANS=100                  T=05
            RF1=0                   RF2=007                 RF3=5                  RF4=5
            RF5=0                   RF6=1                   RF7=0                  RF8=5
            RF9=0                  RF10=0                  RF11=0                 RF12=0
           RF13=0                  RF14=6                  RF15=3                 RF16=0
 GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
 GPRMEM(000004)=000  GPRMEM(000005)=071  GPRMEM(000006)=000  GPRMEM(000007)=000
 WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
 WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
          GPREG=071              WKREG=007               ALUA=071               ALUB=007
         WRYREG=007              BVREG=002              OUTREG=100              INREG=005
         ROMADD=007  ROM(ROMADD)=000000000000           RAMADD=007  RAM(RAMADD)=002

**********************************

      CLOCK TIME=    37          LABEL CYCLE=    37
                                   TRUE LABELS

   /CLK(5)/
          START=0                  RUNS=007                 ANS=100                  T=06
            RF1=0                   RF2=007                 RF3=5                  RF4=5
            RF5=0                   RF6=1                   RF7=0                  RF8=5
            RF9=0                  RF10=0                  RF11=0                 RF12=0
           RF13=0                  RF14=6                  RF15=3                 RF16=0
 GPRMEM(000000)=000  GPRMEM(000001)=000  GPRMEM(000002)=000  GPRMEM(000003)=000
 GPRMEM(000004)=000  GPRMEM(000005)=071  GPRMEM(000006)=000  GPRMEM(000007)=000
 WKRMEM(000000)=000  WKRMEM(000001)=000  WKRMEM(000002)=000  WKRMEM(000003)=000
 WKRMEM(000004)=000  WKRMEM(000005)=007  WKRMEM(000006)=000  WKRMEM(000007)=000
          GPREG=071              WKREG=007               ALUA=071               ALUB=007
         WRYREG=007              BVREG=002              OUTREG=100              INREG=005
         ROMADD=007  ROM(ROMADD)=000000000000           RAMADD=007  RAM(RAMADD)=002

 RUN ENDS AT EOF AFTER    37 ITERATIONS
0END OF INPUT FILE(S). JOB COMPLETE. FILE  5
********* END CDLP *********
```