DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/CSD/TM--198

DE83 012134

A PROGRAMMER'S GUIDE TO FEF: A FAST FRONT-END DATA-ACQUISITION PROGRAM

D. L. Million

Date Published: May 1983

COMPUTER SCIENCES
at
Oak Ridge National Laboratory
Post Office Box X
Oak Ridge, Tennessee 37830

UNION CARBIDE CORPORATION, NUCLEAR DIVISION
operating the
Oak Ridge Gaseous Diffusion Plant . . . Oak Ridge National Laboratory
Oak Ridge Y-12 Plant . . . Paducah Gaseous Diffusion Plant
under Contract No. W-7405-eng-26
for the
DEPARTMENT OF ENERGY

CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT	1
1. INTRODUCTION	3
1.1 Intended Audience	3
1.2 Document Structure	3
1.3 Document Conventions	3
2. HARDWARE ENVIRONMENT	5
2.1 LSI-11/23 Processor	5
2.2 Memory Management	5
2.3 Memory	6
2.4 KVV11-A Programmable Clock	6
2.5 DRV11-B Parallel Interface	8
2.6 BDV11 Bootstrap	8
2.7 DRV11 Parallel Interface	9
2.8 ADAC 1012EX A/D Converters	10
2.9 ADAC 1620DMA Controller	11
2.10 DLV11 Serial Interface	12
3. SOURCE FILE DESCRIPTIONS	13
3.1 SYSDF.MAC	14
3.1.1 Processor Status Word	14
3.1.2 Memory Management Unit	14
3.1.3 Other Peripherals	14
3.1.4 Buffer Sizes	14
3.1.5 Parameter Block Offsets	15
3.1.6 System Global Flags	15
3.1.7 Miscellaneous	15
3.2 LOWCR.MAC	16
3.3 SYSCM.MAC	16
3.3.1 Link Buffers	16
3.3.2 Global Variables	16
3.3.2.1 \$GFLAG	17
3.3.2.2 \$MODE	17
3.3.2.3 \$RDMSK	17
3.3.2.4 \$P2RAT	17
3.3.2.5 \$PINUM	17
3.3.2.6 \$P2NUM	17
3.3.2.7 \$MEMCZ	17
3.3.2.8 \$NSAMS	17
3.3.2.9 \$SAMPL	18
3.3.2.10 \$TRSAM	18
3.3.2.11 \$QTNUM	18
3.3.2.12 \$HTNUM	18
3.3.2.13 \$BPSAM	18
3.3.2.14 \$AVSAM	18
3.3.2.15 \$ADAC Commands	18
3.3.2.16 \$A2BAS	18

3.3.3	Command Table	18
3.3.4	Clock Frequency Table	18
3.4	FESUBS.MAC	19
3.5	INITL.MAC	20
3.6	IDLER.MAC	20
3.7	WTLNK.MAC	20
3.8	START.MAC	21
3.8.1	\$RPARB	21
3.8.2	CHECK	21
3.8.3	INIT	22
3.8.4	START	22
3.8.5	\$RESTR	23
3.9	CLINT.MAC	23
3.9.1	\$CLINO	23
3.9.2	\$CLINI	24
3.9.3	\$CLINT	24
3.9.4	SCAN MODE	24
3.9.5	PT1 Mode	24
3.9.6	PT2 Mode	24
3.9.7	NEXT Sample	25
3.9.8	Clock Recenable	25
3.10	STINT.MAC	25
3.11	ADINT.MAC	25
3.12	TRINT.MAC	26
3.13	XPARB.MAC	26
3.14	XSCAN.MAC	27
3.15	S2ADR.MAC	28
3.16	XTRAN.MAC	28
3.16.1	\$XTRNO	28
3.16.2	\$XTRN	28
3.16.3	XFER	28
3.16.4	\$KILTR	29
3.16.5	DONE	29
3.17	RBOOT.MAC	29
3.18	ERROR.MAC	29
3.18.1	\$FATAL	30
3.18.2	\$NONSI	31
3.18.3	\$LKERR	31
4.	PROGRAM MAINTENANCE	33
4.1	Files	33
4.2	Editing Existing Files	33
4.3	Creating New Files	34
4.4	Source File Assembly	34
4.5	Taskbuilding	35
4.6	Testing and Debugging	35

PROGRAM LISTING

ACKNOWLEDGMENTS

I would like to thank L. R. Layman and S. S. Gould for their help in this project, and Janice Hughes for help in preparing this document. Funding for this project came from the Large Coil Test Facility program of the Fusion Energy Division.

ABSTRACT

The Large Coil Test Facility project of the Fusion Energy Division has a data acquisition system which includes a large host computer and several small, peripheral "front-end" computers. The front-end processors handle details of data acquisition under the control of the host and pass data back to the host for storage. Some of the front ends are known as "fast front ends" and are required to collect a maximum of 64,000 samples each second. This speed and other hardware constraints resulted in a need for a stand-alone, assembly language task which could be downline loaded from the host system into the fast front ends.

FFE (Fast Front End) was written to satisfy this need. It was written in the PDP-11 MACRO-11 assembly language for an LSI-11/23 processor. After the host loads the task into the front end, it controls the data acquisition process with a series of commands and parameters. This Programmer's Guide describes the structure and operation of FFE in detail from a programming point of view. A companion User's Guide provides more information on the use of the program from the host system.

1. INTRODUCTION

The Large Coil Test Facility (LCTF) of the Fusion Energy Division is an experimental facility to test large superconducting magnetic coils of the type which could be used in magnetic containment fusion reactors. The facility includes a data acquisition system which includes a large host computer system (PDP-11/60) and several small front-end systems (LSI-11/23) under the control of the host. The fast front ends are a group of front-end processors intended to each collect up to 64,000 analog samples per second during the coil tests.

FFE (Fast Front End) is a stand-alone data acquisition program which runs in the fast front end processors. The host system downline loads the task into a front end via a communications link and controls all data acquisition functions by issuing commands over this link. The front end, in turn, passes collected data back over the link to the host for storage.

1.1 Intended Audience

FFE was written for the Digital Equipment Corporation (DEC) LSI-11/23 processor, in the MACRO-11 assembly language. This document is directed toward experienced MACRO-11 programmers who need detailed information on the program for the purposes of program maintenance or modification. It assumes that the programmer has a fairly extensive knowledge of both MACRO-11 programming and of the PDP-11/LSI-11 system architecture.

This guide has a companion document, the "User's Guide To FFE". The User's Guide is directed more toward those who wish to write host system programs to interact with FFE, but need not have an intimate knowledge of its structure. The User's Guide contains information on the purposes of the program, downline loading, interprocessor message formats, error handling, and the general algorithm of the program. This material is not repeated in this Programmer's Guide; rather, this guide assumes that the programmer has read the User's Guide and is familiar with it, and understands such things as the command codes and the parameter block.

1.2 Document Structure

Chapter 2 of this Programmer's Guide is devoted to a detailed description of the front-end hardware, highlighting the hardware features that are of most importance in the FFE program. Chapter 3 is a detailed analysis of FFE in the form of a description of each of the MACRO-11 source files in the program, arranged so that the order of the file descriptions follows the flow of program logic. Finally, Chapter 4 provides information on task maintenance, editing the program, rebuilding the task, and debugging.

1.3 Document Conventions

The following conventions are observed throughout this document:

1. The following abbreviations are used. Some of these are common programming abbreviations, some are PDP-11 specific, and some are specific to FFE.

A/D	analog to digital
CSR	control/status register
DMA	direct memory access
KPAR	kernel page address register
KPDR	kernel page descriptor register
kb	kilobyte (1024 bytes)
kw	kiloword (1024 words)
LSB	least significant bit

MMU	memory management unit
MSB	most significant bit
PC	program counter (register 7)
PT1	posttrigger #1
PT2	posttrigger #2
PSW	processor status word
R0-R5	processor registers 0 through 5
RO	read only
R/W	read/write
SP	stack pointer (register 6)
WO	write only

2. All numbers are in decimal radix (base 10), with the exceptions described below.
3. All numbers referring to addresses within the LSI-11/23 memory are in octal radix (base 8).
4. All numbers followed by "(8)" are explicitly defined as octal numbers.
5. In describing bit positions within words, the LSB is bit zero (0), and the MSB is bit 15.
6. I/O page addresses are referred to by their 16-bit addresses, rather than their physical 18-bit addresses (e.g., 172340 rather than 772340).
7. Source files for FFE are referred to by their filenames (e.g., INITL.MAC), whereas routines or subroutines are referred to by their names or address labels (e.g., \$INITL).
8. References to symbolic names within the program follow MACRO-11 addressing conventions. For example, a reference to \$SAMPL refers to the data stored at that address, whereas a reference to #SAMPL refers to the numerical value of the address corresponding to the label "\$SAMPL".
9. The word "sample" refers to one A/D conversion cycle on all of the channels currently being read. Thus, if all 64 channels are active, a sample requires 64 words of memory and "1000 samples per second" implies 64000 A/D conversions per second.

2. HARDWARE ENVIRONMENT

Each of the front-end computer systems is based on the DEC LSI-11/23 processor. In addition, each system contains the following:

1. Memory management unit (standard)
2. Up to 128kw (256kb) of memory
3. KWV11-A (or KWV11-C) programmable real time clock
4. DRV11-B DMA parallel interface
5. BDV11 PROM bootstrap loader
6. DRV11 parallel interface
7. Two ADAC model 1012/1012EX A/D converters
8. Two ADAC model 1620DMA controllers
9. DLV11 serial interface

The remainder of this chapter is devoted to more detailed descriptions of the above hardware. Each description is limited mostly to the information needed to understand FFE's use of the device. For more complete documentation on a particular device, see the appropriate manufacturer's hardware manual.

2.1 LSI-11/23 Processor

The DEC LSI-11/23 processor is a microprocessor which handles the full PDP-11/34 instruction set, including EIS (Extended Instruction Set) instructions. Memory management as a standard feature allows the addressing of up to 128kw (256kb) of memory. To the MACRO-11 programmer, the LSI-11/23 can be treated almost exactly like a PDP-11.

2.2 Memory Management

Since the front-end processors will all likely use more than 32kw of memory, FFE is required to use the Memory Management Unit (MMU). The MMU makes it possible to address 128kw of memory within the constraints of the 32kw (16-bit) program address space, by "relocating" virtual program addresses to physical memory addresses. While a complete discussion of the MMU is beyond the scope of this guide, a short summary of how FFE uses it is appropriate.

The MMU divides the 32kw virtual address space into eight 4kw "pages". Within the virtual address limits of each page, the translation of virtual to physical addresses is controlled by a pair of registers. There are two complete sets of these registers: one for "kernel" mode processor operation, and one for "user" mode. FFE uses kernel mode only, so the set of user registers is not used. Each pair of kernel registers consists of a Kernel Page Address Register (KPAR) and a Kernel Page Descriptor Register (KPDR). The table below shows the addresses of these registers for each page, along with the virtual address range and the physical address range used by FFE for each page.

PAGE NUMBER	KPAR ADDRESS	KPDR ADDRESS	VIRTUAL ADDRESS RANGE	PHYSICAL ADDRESS RANGE
0	172340	172300	000000-017777	000000-017777
1	172342	172302	020000-037777	020000-037777
2	172344	172304	040000-057777	040000-057777
3	172346	172306	060000-077777	060000-077777
4	172350	172310	100000-117777	100000-117777
5	172352	172312	120000-137777	120000-137777
6	172354	172314	140000-157777	varies
7	172356	172316	160000-177777	760000-777777

Note that FFE uses the MMU for an exact correspondence of virtual to physical addresses up to address 137777 (since the top virtual address in the task is approximately 10000, this means that pages 1-5 are never really used). Page 6 is used to access the buffered data in extended memory, so its physical address range varies over the length of the buffer. Page 7 is used for access to the processor I/O page and is thus relocated into the top 4kw of physical memory.

The KPDR's control page size and access. They are not used by FFE other than to specify full access to all pages by the program. The KPAR's are used for the address relocation. A KPAR is loaded with a "relocation bias" specifying the beginning physical address of the page. The relocation bias is equal to the physical address in 32-word blocks, which is equivalent to the physical address divided by 100(8). For example, if FFE needs to access a block of data beginning at physical address 654320, it loads KPAR6 with a value of 6543(8). The first piece of data can then be accessed at virtual address 140020, which is the remainder from the division (654320/100), or the "displacement" from the page base, plus the high bits which specify page 6. The same KPAR6 value can be used until the virtual address goes outside the KPAR6 boundaries, at which point KPAR6 will have to be changed again.

Two other MMU registers are of interest. There are two status registers, SR0 and SR2, at addresses 177572 and 177576. Setting bit 0 in SR0 turns on the MMU; i.e., after the bit is set, the MMU begins relocating all addresses based on the contents of the KPAR's. Otherwise, the status registers are not used by FFE except for reporting MMU errors. If an MMU error occurs, a trap to vector address 250 is generated.

2.3 Memory

The LSI-11/23 can address up to 128kw of memory. However, as with all PDP-11's, the top 4kw of address space is reserved for the I/O page, so the maximum user memory is 124kw. Less than 2kw is required for the FFE task; the remainder is all used to buffer data. FFE is not written to use any set amount of memory; instead, the program itself determines how much memory is available to it (see description of INITL.MAC in Chapter 3). Thus, if the amount of memory in the processor changes, FFE can still be used without modification. The advantage to having larger amounts of memory is, of course, that more data can be buffered during data acquisition.

2.4 KWV11-A Programmable Clock

The KWV11-A programmable real time clock is used to control the timing of all data acquisition processes by FFE. It can be programmed to generate interrupts at intervals ranging from 1 microsecond to over 1000 seconds; however, FFE does not use any interval shorter than 1 millisecond (1/1000 second). The KWV11-A has two registers available to the programmer and two interrupt vectors, as follows:

ADDRESS	DESCRIPTION
170420	CSR
170422	Buffer/preset register (BPR)
440	Clock vector
444	Schmitt trigger 2 (ST2) vector

The following bit definitions apply to the CSR:

Bit	Access	Description
15	R/W	ST2 flag--set by the firing of Schmitt trigger 2. If bit 14 is set, an interrupt is generated at the ST2 vector; if bit 13 is set, this causes the clock to start running by setting bit 0. Cleared under program control. If ST2 fires again while this bit is still set, bit 12 is asserted.
14	R/W	ST2 interrupt enable--when set, the assertion of bit 15 will cause an interrupt at the ST2 vector.
13	R/W	ST2 GO enable--when set, the assertion of bit 15 will set the GO bit (bit 0), starting the clock, and bit 13 will be cleared.
12	R/W	Flag overrun--normally clear, this is set by the clock if an overflow occurs while bit 7 is still set (which generally means that the program is not keeping up with the speed of the clock), or when ST2 fires while bit 15 is still set (meaning that an ST2 interrupt has been lost).
11-8		Not used.
7	R/W	Overflow flag--set each time the clock counter overflows. Causes an interrupt if bit 6 is set. Must be cleared by the program before another overflow occurs; if not, bit 12 (indicating an overrun error) is set.
6	R/W	Overflow interrupt enable--when set, an overflow (assertion of bit 7) causes an interrupt at the clock overflow vector.
5-3	R/W	Rate--3 bits that select the clock counting frequency as follows:

5	4	3	RATE
0	0	0	Stopped
0	0	1	1 megahertz
0	1	0	100 kilohertz
0	1	1	10 kilohertz
1	0	0	1 kilohertz
1	0	1	100 hertz
1	1	0	ST1 (external oscillator)
1	1	1	Line (60 hertz)

2-1	R/W	Clock mode--There are four clock modes, but the only one used by FFE is mode 1 (bit 2 clear, bit 1 set), or "repeat overflow" mode (see below).
0	R/W	GO--STARTS the clock counting.

The KWV11-A can be used in several different ways, but FFE uses it in only one mode, called "repeat overflow" mode. The clock operates by counting "ticks", which are provided by an internal or external oscillator running at a known frequency. The BPR is loaded with the 2's complement of the desired "overflow count", i.e., the number of ticks desired between interrupts. At the start of each cycle, the clock moves the contents of the BPR to an internal counting register. Then, at each clock tick, this internal register is incremented. When it overflows (reaches zero), the overflow bit (bit 7) is set and the clock immediately reloads the counting register and begins counting again.

The contents of the BPR remain unchanged unless modified by the program. If the overflow count must be changed while the clock is running, a new value is loaded into the BPR. This updated count is used for the first time after the next clock overflow, when the counter is reloaded from the BPR. The BPR can be treated as an unsigned 16-bit integer; for example, if 60000 ticks are desired between overflows, the 2's complement of this number, which is +5536, can be loaded into the BPR.

2.5 DRV11-B Parallel Interface

The DRV11-B parallel interface provides the communications link between the front end and the host computer. The comparable device in the host PDP-11 is a DR11-B, which is device XB: to the RSX-11M operating system. FFE does not manipulate the DRV11-B directly, but instead uses a package of link subroutines called FESUBS, which was already available when FFE was written (see further information on FESUBS in Chapter 3). Because of this, the DRV11-B device is not described in detail here.

2.6 BDV11 Bootstrap

The BDV11 PROM bootstrap loader provides means for stopping, starting, and rebooting the LSI-11/23. It also contains 2kw of start-up diagnostics in ROM (Read Only Memory), and provides up to 2kw additional PROM (Programmable Read Only Memory) for user start-up routines. In the LCTF front end, the BDV11 PROM contains a short program which prepares the processor to receive a downline loaded task from the PDP-11/60 DLN utility (see the User's Guide for details on running DLN). Thus, the processor is ready for a downline load immediately after being booted.

Before downline loading a user's task, DLN first transfers another task called LSILDR (see the Chapter 3 section on FESUBS.MAC for more information on LSILDR). LSILDR is a more sophisticated link receiver program than the tiny PROM routine. Therefore, the PROM routine is set up specifically to receive LSILDR.TSK and start running it. LSILDR.TSK is always taskbuilt to have a starting address of 154000 and is short enough to transfer in one message. The PROM routine for receiving it, as it would be represented in MACRO-11 assembly language, is listed below.

```

START:  BIT    #4000,@#172414 ; IS 11/60 READY?
        BEQ    START          ; LOOP UNTIL IT IS
        MOV    #172410,R0      ; GET DRV11-B WORD COUNT ADDRESS
        MOV    6(R0),(R0)      ; LOAD WORD COUNT
        NEG    (R0)+           ; NEGATE IT
        MOV    #154000,(R0)+   ; SET LOADING ADDRESS
        MOV    #5,(R0)         ; START THE DRV11-B
15:     TSTB    (R0)            ; IS READY BIT ON YET?
        BPL    15              ; LOOP UNTIL READY
        JMP    @#154000        ; JUMP TO LSILDR ENTRY

```

The FROM routine first checks the DRV11-B CSR until it sees a particular bit set from the PDP-11/60 end of the link. At that point, it retrieves the word count of the coming message from the DRV11-B data register (where it was written by the PDP-11/60), loads the word count into the DRV11-B word count register, and negates it (as is required by the hardware). The starting address of LSILDR is loaded into the DRV11-B bus address register and the transfer is initiated by loading the DRV11-B CSR with the proper bits. The routine then loops until the "ready" bit comes on in the CSR, after which it jumps into the LSILDR entry point.

The BDV11 can be used to bootstrap the LSI-11/23 from switches in the rear of the processor, or it can be accessed by a program in the LSI-11/23 to do an online reboot. To do an online reboot, a register in the BDV11, called the Page Control Register (PCR), must be cleared to set the internal BDV11 program back to the beginning. The address of the PCR is 17757. Then, a jump to the bootstrap address of 173000 must be executed.

2.7 DRV11 Parallel Interface

The DRV11 parallel interface provides parallel communication with an external device, and two interrupt inputs to the LSI-11/23. In the LCTF front end, the DRV11 interrupt inputs are used to detect the "quench" or "heater pulse" transient events. The DRV11 has three registers available to the programmer and two interrupt vectors, as follows:

ADDRESS	DESCRIPTION
167770	CSR
167772	Output buffer
167774	Input buffer
300	Interrupt vector "A"
304	Interrupt vector "B"

The following bit definitions apply to the CSR:

BIT	ACCESS	DESCRIPTION
15	RO	Request "B"--set or cleared by external device. In front end, set by "heater pulse" trigger.
14-8		Not used.
7	RO	Request "A"--set or cleared by external device. In front end, set by "quench" trigger.
6	R/W	Interrupt enable "A"--when set by the program, the assertion of bit 7 by an external device will generate an interrupt at the interrupt "A" vector.
5	R/W	Interrupt enable "B"--when set by the program, the assertion of bit 15 by an external device will generate an interrupt at the interrupt "B" vector.
4-0		Not used.

The input buffer register is not currently used by FFE. The output buffer can be used to signal the host of certain conditions. Currently, only one bit (bit 0) of the output buffer is used to interrupt the host upon the conclusion of transient data collection or upon the detection of a fatal error in FFE. To provide an interrupt to the host, bit 0 must be set and then cleared.

2.8 ADAC 1012EX A/D Converters

The ADAC model 1012EX is a data acquisition device which allows analog to digital (A/D) conversion on up to 64 single ended, or 32 fully differential, input channels. There are two of these devices in each of the front ends. The particular configuration used in the LSI-11/23's is as follows:

1. 32 fully differential input channels per device (for a total of 64 fully differential channels),
2. 100 kilohertz sample throughput,
3. Voltage input range -10VDC to +10VDC,
4. Programmable gains (only gain 1 is used by FFE),
5. 12-bit digital output--sign extend for negative values not provided.

The relevant device registers and interrupt vectors are listed in the following table.

ADDRESS	DESCRIPTION
176770	CSR #1
176772	Data buffer register #1
177000	CSR #2
177002	Data buffer register #2
130	Interrupt vector #1
140	Interrupt vector #2

The following bit definitions apply to the CSR's:

BIT	ACCESS	DESCRIPTION
15-14		Not used by FFE.
13-8	R/W	Channel number--these six bits form the number of the channel currently being converted. Possible values are 0-63.
7	RO	Done--set at the end of a data conversion. Cleared by reading the data buffer register, system INIT, or by writing into the CSR.
6	R/W	Done interrupt enable--when set, the assertion of bit 7 causes an interrupt. These interrupts are not used by FFE.
5		Not used.
4-3	R/W	Gain bits--FFE uses only a gain of 1, which requires that both of these bits always be set.
2	R/W	Sequential enable--when set, the channel bits (8-13) will increment to the next channel automatically at the end of a conversion.
1	R/W	External enable--when set, allows an external signal to start a conversion.
0	WO	Start--setting this bit starts a conversion.

The data buffer register contains the 12-bit result after the end of a conversion. See the following section for information on how this device interacts with the ADAC 1620DMA.

2.9 ADAC 1620DMA Controller

The ADAC model 1620DMA controller is used in conjunction with the ADAC 1012EX A/D converter to provide autosequenced direct memory access (DMA) A/D data acquisition. It allows the user to specify a starting channel number, the number of samples to collect, and the address of the user's data buffer. The 1620DMA then commands the 1012EX to convert the required number of samples in the correct order, and it returns the results directly to the desired location in memory without the intervention of the LSI-11/23 processor. The front ends each have two of these devices, one to control each of the 1012EX converters. The device registers and interrupt vectors for these devices are as follows:

ADDRESS	DESCRIPTION
172420	1620DMA #1 word count register
172422	1620DMA #1 bus address register
172424	1620DMA #1 CSR
172426	1620DMA #1 comparator register
172440	1620DMA #2 word count register
172442	1620DMA #2 bus address register
172444	1620DMA #2 CSR
172446	1620DMA #2 comparator register
34	1620DMA #1 interrupt vector
144	1620DMA #2 interrupt vector

Before the start of a DMA conversion operation, the word count register is loaded with the 2's complement of the number of samples, or words, desired. The bus address register is loaded with the lowest 16 bits of the physical address where the results are to be stored. The comparator register is used to reset the ADAC 1012EX channel number to zero when multiple sampling of channels is being done; FFE does not use this feature. The following bit definitions apply to the CSR's:

BIT	ACCESS	DESCRIPTION
15	RO	Error--becomes set if bits 14 or 15 are set. Causes bit 7 to be set, and causes an interrupt if bit 6 is set.
14	R/W	Nonexistent memory--becomes set if a DMA transfer is attempted to a nonexistent address. Causes bit 15 to be set. Cleared by program.
13	RO	Attention--becomes set via an external device, if desired. Not used on front-end processors.
12-8		Not used.
7	RO	Ready--cleared by setting bit 0. Set by word counter overflow when all DMA cycles completed, or set when bit 15 is set.

6	R/W	Interrupt enable—when set, the assertion of bit 7 will generate an interrupt at the device vector.
5-4	R/W	Extended address bits—these are the high order bits (bits 16 and 17) of the 18-bit physical address of the DMA buffer. The low order 16 bits are loaded into the bus address register.
3-1		Not used.
0	WO	Go—initiates a DMA conversion operation (but does not actually start conversion—see below).

FFE uses the ADAC devices to perform A/D conversion on a sequence of channels (not repeating any of them), at a gain of 1 and with interrupt control. The actions necessary to accomplish this are as follows:

1. Load the 1620DMA comparator register with a value of 63, which is the highest possible channel number (on 0-63 basis). This effectively disables the use of this register, and it need not be done more than once.
2. Read the 1012EX data buffer register (a TST instruction will suffice) to clear the 1012EX "done" bit (bit 7).
3. Load the 1620DMA word count register with the 2's complement of the number of channels to convert.
4. Calculate the 18-bit physical address where the results are to be stored. There must be enough room between this address and the top of memory to store all of the results requested. Load the low order 16 bits into the 1620DMA bus address register.
5. Shift the high order address bits four places to the left, which will put them in the correct positions of the extended address bits in the 1620DMA CSR (bits 4 and 5). OR in the "go" bit (bit 0) and the interrupt enable bit (bit 6), and load the entire result into the 1620DMA CSR. This step does NOT start the conversion.
6. Load a register with the lowest channel number desired (lowest possible channel number is 0). Swap this value to the high byte. OR in both gain bits (bits 3 and 4), the sequential enable bit (bit 2), the external enable bit (bit 1), and the "start" bit (bit 0). Load the entire result into the 1012EX CSR. This initiates the first A/D conversion in the sequence.

An interrupt will be generated at the 1620DMA vector when the last result has been deposited in memory. The 1012EX sequential enable and external enable bits are necessary to allow the 1620DMA to control the 1012EX during the conversion process.

2.10 DLV11 Serial Interface

Each of the LSI-11/23 processors includes a DLV11 serial interface, which is normally used to connect a console terminal to the system. It is thus possible to connect a terminal to a processor for debugging purposes, but under normal operating conditions there will be no terminals on the front ends.

3. SOURCE FILE DESCRIPTIONS

This chapter contains detailed descriptions of all of the MACRO-11 source files used in FFE. The program is broken up into a number of source files for reasons of modularity and ease of editing. They are described in a sequence which is meant to help the programmer understand the flow of the program. The sources are stored under account [160,160] on the PDP-11/60 system disk. The file names, in the order in which they are described, are as follows:

SYSDF.MAC	System definitions
LOWCR.MAC	Low memory layout
SYSCM.MAC	System common data storage area
FESUBS.MAC	Link subroutines
INITL.MAC	Task initialization routines
IDLER.MAC	Task link/idle loop
WTLNK.MAC	Link completion wait routine
START.MAC	Scanning start-up routines
CLINT.MAC	Clock interrupt handler
STINT.MAC	Clock ST2 interrupt handler
ADINT.MAC	ADAC 1620DMA interrupt handlers
TRINT.MAC	Transient interrupt handler
XPARB.MAC	Transmit parameter block routine
XSCAN.MAC	Transmit scan averages routine
S2ADR.MAC	Sample to address conversion routine
XTRAN.MAC	Transmit transient data routine
RBOOT.MAC	Reboot routine
ERROR.MAC	Fatal error handler

A number of conventions are followed throughout the source files, which the programmer is urged to adhere to in any program modifications (an exception is FESUBS.MAC, since it was a previously written package and was not written as part of FFE). Many of them are identical or similar to conventions used in RSX-11M system software. A list of these conventions appears below.

1. All symbols of the form "\$XXXXX", where the X's stand for any alphanumeric characters, refer to global addresses within the program.
2. Constants for values such as device register addresses are represented by alphanumeric names without the "\$" character.
3. Symbols of the form "X.xxxx" refer to offset constants; i.e., they can be used as symbolic offsets to a base address for accessing tabular data.
4. Symbols of the form "XX.XXX" refer to bit mask constants, which are used for setting, clearing, or testing one or more bits of interest within a word.
5. Each file begins with a ".TITLE" line and an ".IDENT" line. The ".IDENT" string consists of the initials of the last person to edit the file, followed by the edit number. This should be changed whenever the file is edited.
6. Each file contains the same header comment lines, which state that the file is part of the LCTF fast front end stand-alone task.

7. Following the header lines is a group of comments giving the creation date and all edit dates, and the persons responsible. Each time the file is edited, a new edit line should be added here, possibly followed by a short description of what was changed.
8. Following the edit level lines is a short description of the contents of the file.
9. Each instruction contains a comment field, except where the meaning is obvious due to repetition, etc. In addition, comment lines are interspersed in the code where appropriate to better explain program intentions or to mark off distinct sections of code.
10. Assembly constants are used in place of numerical constants wherever possible, to avoid the proliferation of "magic numbers".
11. The "CALL" macro is used in place of all "JSR PC" instructions, and the "RETURN" macro in place of all "RTS PC" instructions.

The descriptions of the source files begin below. It is recommended that the reader have a printed copy of the sources for reference during the reading of this chapter.

3.1 SYSDF.MAC

This file is an assembler prefix file containing all of the "system definitions" of assembly constants. It must be used as a prefix to the assembly of all of the other source files (except FESUBS.MAC). The symbols are divided up into groups as shown below.

3.1.1 Processor Status Word

These symbols include the processor priority levels, and bit masks for the previous mode and current mode bits. Loading one of the priority constants into the PSW sets the processor to that priority level (these symbols match the ones used in RSX-11M). The mode constants are not currently used.

3.1.2 Memory Management Unit

These symbols include the addresses of the MMU page address and page descriptor registers, the MMU status registers, and the trap vector. Bit masks are defined for the PDR page access and page size control bits, and a combined bit mask for an upward expandable, full size, R/W page, which is the only type of page currently used by FFE.

3.1.3 Other Peripherals

The device registers and interrupt vectors are defined for the KWV11-A clock, the DRV11 parallel interface, the ADAC 1012EX and 1620DMA data acquisition devices, and the BDV11 bootstrap. Bit mask constants are defined for the bits of interest in all of the device CSR's (since the DRV11-B link device is completely handled by FESUBS, it does not appear). In addition, a combined bit mask is defined for the ADAC 1012EX, combining the gain=1 bits, the sequential enable bit, the external enable bit, and the start bit. These are the bits used by FFE in controlling this device.

3.1.4 Buffer Sizes

The sizes of various data buffers used in FFE are defined. The link transmit buffer is used to assemble messages transmitted to the host, and the receive buffer is used to store received messages. The parameter buffer stores the current parameter block data. The error buffer is a special buffer used for dumping diagnostic data, which are transmitted to the host after a fatal system error. If the sizes of any of these buffers need to be changed, changing the appropriate symbol value is the only step necessary to do so.

3.1.5 Parameter Block Offsets

These offset symbols are used to access specific parameters in the parameter block. The symbol is added to the base address of the parameter block, either directly or via index mode addressing, to find the parameter described by the symbol. The programmer must note that these symbols may not be shuffled around at will. Their order must agree with that sent by the host, of course, but FFE also relies on the parameters being in a certain sequence. Changing them around would entail modification of several other program modules.

3.1.6 System Global Flags

FFE includes a global flag word, and these bit mask symbols are used to set and clear flags within the flag word. The flags are used by many program modules to keep track of the current state of the task. The global flags are as follows:

FLAG	DESCRIPTION
GF.ACC	Data access flag. When this flag is set, the scan average routine (see XSCAN.MAC) is in the process of gathering data from the ring buffer to be used in calculating scan averages. The flag is checked by the clock interrupt code (see CLINT.MAC) to see whether or not it needs to check for data overrun.
GF.AVO	Average overrun. If GF.ACC is set, the clock interrupt handler checks to see if it has overrun the data being averaged. If it has, it sets GF.AVO to inform the scan average routine that this has occurred.
GF.EOT	End of transient. This flag is set after transient data recording has finished. It is cleared when scan data acquisition is restarted. Retrieval of transient data is permitted while this flag is set.
GF.FTL	Fatal error. This flag is set if the system detects a fatal error.
GF.PAR	Parameter block received. This flag is set when FFE receives the first parameter block to note that at least one such block has been received.
GF.RD1	Ready #1. This flag is cleared by the clock handler when an ADAC A/D conversion is initiated on ADAC unit #1. The flag is set when ADAC 1620DMA #1 interrupts when all its conversions are finished (see ADINT.MAC). At the next clock interrupt, the clock handler checks the flag to make sure that the previous sample was finished.
GF.RD2	Ready #2. Same as GF.RD1, except for ADAC unit #2.
GF.SDA	Scan data available. This flag is set when scan data acquisition is started, and after enough samples are collected to enable a scan average to be done. For example, if the parameter block specifies 10 samples per average, this flag will be set after 10 samples are collected. The flag is checked by the scan average routine to see if it may calculate an average.

3.1.7 Miscellaneous

These symbols include symbolic representations of the interrupt number passed back to the host in the parameter block, the clock operation flag, the high interrupt vector in the system, and the desired system stack size, in words.

3.2 LOWER.MAC

This file defines the "low core" or the lowest part of the processor memory. It contains no executable code; it merely defines the contents of certain memory locations. Since it is written assuming that location zero of the assembly module is equivalent to location zero in physical memory, it must be the first module in the list of object modules linked by the task builder when FFE is built.

Initially, the file fills the low memory up to the high vector with vectors pointing to the "nonsense interrupt" entry point (see ERROR.MAC). This forces all unexpected interrupts to immediately return. Next, the processor trap vectors are filled in. The power fail trap is directed to the nonsense interrupt (since a power failure should actually return the processor to console ODT mode). The IOT trap is used for processing link errors, and thus is directed to \$LKERR. The EMT trap is used for fatal system errors and is directed to \$FATAL. Both of these addresses are in ERROR.MAC. The other processor traps are directed to \$FATAL, but they include unique bit settings in the PSW portion of the vector which allow the error processor to distinguish the traps from one another. Note that all traps and interrupt vectors are set up for processor priority 7 upon entry (for faster response), except for the link error trap. Since the link error code is rather lengthy and could interfere with data acquisition activities, it is allowed to run at priority zero (fully interruptible).

Finally, the device interrupt vectors are defined. Each device interrupt vector is directed toward the appropriate interrupt handler code. The DRV11 transient interrupts are both directed to the same address, but again the low byte of the PSW is used to distinguish the two. The MMU vector is directed to the error processor, since it only interrupts on an addressing error of some sort. After all the interrupt vectors are defined, the top of the system stack is defined as \$STACK.

3.3 SYSCML.MAC

This file contains the "system common" data storage areas, which are used by all of the other modules in the program. It consists solely of blocks of words and global addresses, and contains no executable code.

3.3.1 Link Buffers

First to appear are the FESUBS argument lists. These are necessary for the use of the FORTRAN callable FESUBS routines. For a FORTRAN callable routine to be called from a MACRO-11 module, an argument block must be defined which consists of (a) one word containing the number of arguments, and (b) a list of argument addresses. For example, the FEXMIT subroutine is called from FORTRAN as follows:

```
CALL FEXMIT(DATBUF,WDCNT)
```

where DATBUF is an integer array containing the message, and WDCNT is the length of the message, in words. Translating this into a MACRO-11 argument block called \$XARG, the block consists of the value 2 (for the number of arguments), the address of the transmit buffer (\$XBUF), and the address where the message length is stored (\$XSIZ). The receive argument block \$RARG, containing \$RBUF and \$RSIZ, is similar.

Below the argument blocks appear the message size words; the actual message sizes are defined in SYSDF.MAC. Then the data buffers themselves appear; \$RBUF is the link receive buffer, \$PBUF is the parameter block storage buffer, and \$XBUF is the link transmit buffer. \$EBUF is the fatal error buffer used to transmit error information to the host. It is initialized with the fatal error code IE.FHE, and the remainder filled with -1's.

3.3.2 Global Variables

The global variables consist of all of the one-word variables used globally by the program. This area directly follows the error buffer, so that upon a fatal error, the global variables are transmitted to the host along with the error buffer. Each of the variables is described in detail below.

3.3.2.1 **\$GFLAG** — Global flag--the flags defined in **SYSDF.MAC** are stored here as bit values.

3.3.2.2 **\$MODE** — Mode--the current program operating mode. The mode is used to keep track of the status of the program and to determine which host commands are legal or illegal at any one time. The possible modes are:

MODE	DESCRIPTION
-2	Prescan mode--FFE is doing nothing in particular, other than waiting for a command to begin scanning. FFE starts out in prescan mode and returns to prescan mode after a transient (i.e., after all of the transient data have been retrieved by the host).
-1	Scan mode--the system is performing scan data acquisition. In this mode, scan average data are available, if enough time has been allowed to collect sufficient samples.
0	Posttrigger #1 mode--a transient interrupt has been sensed and FFE is collecting primary posttrigger data. Requests for scan average data are rejected at this and higher modes.
1	Posttrigger #2 mode--the primary posttrigger phase of a transient has finished and FFE is now collecting secondary posttrigger data.
2	Posttransient mode--all posttrigger data have been collected, and the system is waiting to send (or is in the middle of sending) the transient data to the host.

3.3.2.3 **\$RDMSK** — Ready mask--this word, during data acquisition, contains the bit values of the "ready" global flags (see **SYSDF.MAC**) for the active ADAC units (#1, #2, or both). This is used to clear the "ready" flags only for the units which are actually running.

3.3.2.4 **\$P2RAT** — Posttrigger #2 rate--contains the value of the pt2 clock rate during a transient, so that it can be accessed quickly when the time comes to switch to that mode.

3.3.2.5 **\$PINUM** — Posttrigger #1 number--contains the number of pt1 samples to be collected during a transient. The clock uses this word to count down the number of samples to zero.

3.3.2.6 **\$P2NUM** — Posttrigger #2 number--contains the number of pt2 samples and is used in a similar manner to **\$PINUM**.

3.3.2.7 **\$MEMSZ** — Memory size--after the initialization code (see **INITL.MAC**) determines the total amount of memory in the system, it is stored here as a number of 32-word blocks.

3.3.2.8 **\$NSAMS** — Number of samples--FFE works on the basis of "sample number"; i.e., when it is starting a scan, it divides the entire memory available to the ring buffer into "samples", each of a length in words equal to the number of channels being scanned. It uses this sample number to keep track of the ring buffer position. **\$NSAMS** is this total number of samples.

NOTE: If only one channel is being scanned, and if the ring buffer is large enough, the total number of samples could exceed 65535 (e.g., a 16-bit word). In this case, the total number of samples is limited to 65535, and only a part of the available ring buffer is used.

3.3.2.9 **\$SAMPL** — Sample number—contains the current sample number. It starts at zero and is incremented until it becomes equal to \$NSAMS, at which point it goes back to zero again.

3.3.2.10 **\$TRSAM** — Transient sample number—this word is used to note the current position in the ring buffer (via sample number) at the time that a transient interrupt occurred.

3.3.2.11 **\$QTNUM** — Quench transient number—equal to the total of pretrigger, pt1, and pt2 samples passed in the parameter block for the quench interrupt. Must be smaller than \$NSAMS.

3.3.2.12 **\$HTNUM** — Heater transient number—similar to \$QTNUM, except it is for the heater interrupt.

3.3.2.13 **\$BPSAM** — Bytes per sample—used in buffer calculations, equal to the number of channels being scanned multiplied by two.

3.3.2.14 **\$AVSAM** — Average sample—during a scan average operation, this word is loaded with the sample number where the averaging routine is accessing data. This is checked by the clock interrupt code (see CLINT.MAC), and if the current sample number (\$SAMPL) becomes equal to \$AVSAM, a data overrun error condition is reported.

3.3.2.15 **ADAC Commands** — The labels of the form \$ADNXX where N is either 1 or 2, and XX is a pair of letters, refer to the ADAC device command words. The 1 refers to ADAC unit #1, and the 2 to unit #2. The different command words are:

AD	--	1012EX A/D device CSR
DM	--	1620DMA device CSR
WC	--	1620DMA word count register
BA	--	1620DMA bus address register

The command words are all calculated upon scanning start up, and the resultant values stored in these words for use during data acquisition.

3.3.2.16 **\$A2BAS** — ADAC #2 base—contains the base buffer address for ADAC unit #2. Unit #1 uses the bottom of the ring buffer as its base address, but unit #2, of course, must be offset from this by some amount dependent upon the number of channels being collected by unit #1. This produces a slightly different base address for unit #2, and this is stored in \$A2BAS.

3.3.3 Command Table

\$CMDTB is a table of subroutine addresses to handle the various link commands. A command received from the host is used as an offset into this table, to find the appropriate subroutine address for the command. A call to that subroutine is then executed. Any new commands added to the program will require a new entry in this table. The global variable KMAX at the bottom of the table is equal to the maximum legal command code, which is used for checking the commands from the host.

3.3.4 Clock Frequency Table

The clock frequency parameter sent from the host is based upon the clock rate bits in the KWV11-A CSR. Multiplying the clock frequency parameter by 10(8) produces the correct rate bits to go into the clock CSR. This table, at \$FRQTB, is used to determine which clock frequency commands

are legal, and to determine the validity of the clock rate (tick) parameters. A negative 1 in the table denotes an illegal value for the frequency parameter (e.g., "0" is illegal since this is for a stopped clock). A positive value means that the frequency is legal, and this positive value is the minimum clock tick count which can be used at that frequency. Currently, these values are chosen to limit the sampling rates to 1000 samples per second, maximum.

3.4 FESUBS.MAC

This is the package of DRV11-B link subroutines, written by J.S. Goddard, which was incorporated into FFE to handle the link communications. Since it was not written as part of FFE, it will not be described other than to show how it is used by FFE. The copy of FESUBS.MAC used in FFE was modified slightly, to remove all .PSECT directives from the source code. This was done to force the task builder to maintain the desired sequence of modules in the task image (see Chapter 4).

FESUBS is used by the DLN utility to load a task image into the front end. The small BDV11 bootstrap program is used to pass a task called LSILDR (LSI Loader) into the front end. This task then establishes a more sophisticated link, using FESUBS, to receive the task image which is actually to run in the front end (in this case, FFE). When LSILDR is finished receiving the task image, it executes a jump instruction to the task transfer address, which begins execution of the loaded task.

When FFE starts running, it is concerned with four subroutines (FEINIT, FERECV, FEXMIT, and FETERM) and two status words (LNKIO and LNKERR) within FESUBS. The use of these is outlined below:

A. Initializing the link:

1. Execute CALL FEINIT.
2. No errors are possible.

B. Receiving a message:

1. Move #SRARG (receive argument block address) to R5.
2. Execute CALL FERECV.
3. Wait until LNKIO is equal to +1.
4. Test LNKERR; if zero, no error; if nonzero, a link error has occurred.

C. Transmitting a message:

1. Move #XARG (transmit argument block address) to R5.
2. Execute CALL FEXMIT.
3. Wait until LNKIO is equal to +1.
4. Test LNKERR as before.

D. Terminating the link:

1. Execute CALL FETERM.
2. Test LNKERR; if nonzero, a link hardware error has occurred, otherwise all errors have cleared.

3.5 INITL.MAC

This file contains the program entry point and the system initialization code. It contains two important global symbols; \$INITL, which is the task transfer address (where the program begins execution), and \$BUFFR, the bottom address of the ring buffer. It can be seen from the file that the two symbols have the same address, which means that the initialization code is covered up with data after it is used. This is possible since \$INITL is not used more than once, and it increases the size of the ring buffer. To prevent this situation from corrupting other code modules, INITL.MAC must be linked last of all by the task builder (see Chapter 4).

Upon entry from LSILDR (see above), the module first does a RESET instruction to initialize all of the hardware, locks out interrupts by putting the processor at priority 7, and sets the stack pointer register to \$STACK (see LOWCR.MAC). Since LSILDR modifies the EMT trap vector at address 30, INITL resets it to the proper FFE vector.

The module then sets up the MMU. All KPAR's from 0 to 5 are set for an exact correspondence of virtual to physical address. KPAR6 is skipped. KPAR7 is set to access the I/O page. All of the KPDR's are set for full page, upward expandable, R/W access. Then bit 0 of status register 0 is set to turn on the MMU. After this point, all address references are processed by the MMU.

The next step is location of the top of physical memory. If any nonexistent memory is accessed, the processor will trap to vector 4. This vector is made to point to a special trap handler for the purposes of the memory test. The trap handler sets the C-bit in the stacked PSW, and returns; thus, after an instruction which causes a trap, the PSW C-bit will be set. The memory test routine uses KPAR6 to step through memory in 1kw increments, performing a TST instruction at each step until either a trap is detected (via the C-bit) or the maximum possible address is reached (760000). At this point, the top of memory, in 32-word units (equivalent to a MMU relocation bias), is stored in \$MEMSZ.

A similar trap is then set up for a device test. All of the peripheral devices are tested by accessing one of the registers in each one. If a trap occurs, the device is not online, or perhaps at the wrong address. If this occurs, the trap handler puts the offending addresses in the error buffer, and a fatal error is declared (see ERROR.MAC).

To finish up, INITL sets both ADAC 1620DMA comparator registers to the highest possible channel numbers; this effectively disables the use of those registers (see Chapter 2). The correct trap 4 vector is restored, and the link is initialized by calling FEINIT. The processor priority is dropped to enable interrupts, and finally a jump to \$IDLER, the link idle loop, is executed.

3.6 IDLER.MAC

This file contains the link idle loop \$IDLER. When the system has nothing else to do, it is somewhere in this loop. All traps and interrupts return here (or to one of the subroutines called from here). It first calls FERECV to receive a link message. It then calls \$WTLNK (see WTLNK.MAC) to wait until the receive request has been satisfied. If the command word in the receive block is illegal (i.e., less than or equal to zero, or too large), an error is returned to the host immediately; otherwise, the command is converted to a \$CMDTB offset (see SYSCM.MAC), and the appropriate subroutine from the table is called.

If the command was to reboot the system, the subroutine does not return since the system is booted immediately; otherwise, the subroutine places reply data into \$XBUF and returns. IDLER then calls FEXMIT to send the reply message. \$WTLNK is again called to wait for completion, and the routine then skips back to the beginning to start the loop over again.

3.7 WTLNK.MAC

This file contains \$WTLNK, which is called by IDLER.MAC to wait for link message completion. The routine executes a WAIT instruction, which causes the processor to pause until any interrupt occurs. Upon the occurrence of an interrupt, WTLNK checks LNKIO to see if the interrupt was from the link. If LNKIO is still nonpositive, the WAIT is executed again; otherwise, LNKERR is tested for

link errors. If LNKERR is zero, no error occurred and the subroutine returns. If it is nonzero, an IOT is executed, which traps to SLKERR, the link error handler (see ERROR.MAC). If the program returns from the trap, the link has been reinitialized, so WTLNK changes its return address (on the stack) so that it returns to the link receive step of IDLER.MAC, no matter what it was doing when the error occurred. This means that if a recoverable link error occurs, the host can attempt recovery by reinitializing the host end of the link and then retrying the last transmission.

3.8 START.MAC

This file contains two of the command table subroutines called by IDLER.MAC. They are concerned with receiving the parameter block and starting scan data acquisition. To FFE, receiving a parameter block implies a new scan start-up with the new parameters.

3.8.1 SRPARB

The subroutine SRPARB (Receive Parameter Block) is called to process host command 1. First, it checks \$MODE (see SYSCM.MAC). If it is negative, it is all right to start scanning; otherwise a transient is in progress and the subroutine returns with a "transient in progress" status code in word 1 of \$XBUF. Next, it begins checking parameters for validity.

3.8.2 CHECK

This local label marks the parameter checking section of the subroutine. The following steps are taken in this section:

1. The clock frequency parameter is checked for legal values using the clock frequency table (see SYSCM.MAC).
2. The starting channel number and the number of channels are checked to make sure no channel number outside the range 1-64 is possible.
3. The number of samples available in the ring buffer, based upon the channel count parameter and the total ring buffer size, is calculated. If the result is greater than 16 bits, the sample count is limited to the highest 16-bit value (65535).
4. All clock rates (tick counts) are checked to see that the data collection rate does not exceed 1000 samples per second. This limit was chosen arbitrarily as being probably close to the physical limitations of the hardware.
5. The transient interrupt sample counts (pretrigger, primary posttrigger, and secondary posttrigger) are added together for each type of transient, and the total is checked to make sure it does not exceed the sample count calculated above. Also, each of the individual sample counts must be at least 1.
6. The number of scans per average is checked to make sure it is a nonzero, positive value.
7. The clock control flag is checked to see if it is equal to one of the two legal values.

If any of these checks fail, the subroutine returns with a "bad parameter" error in word 1 of \$XBUF. If a previously started scan was already going on, it has not been affected up to this point. Note that the CHECK routine proceeds through the list of parameters assuming that they are in a particular order (see SYSCM.MAC). If the parameters are rearranged, this code will have to be modified.

3.8.3 INIT

If all parameters are valid, this section is entered, and any previously started scanning is aborted. This section performs the following:

1. The clock and the DRV11 interface are disabled to prevent interrupts during initialization.
2. \$MODE is set back to prescan mode, and the "scan data available" flag (GF.SDA) is cleared.
3. The global flag GF.PAR is set to indicate that at least one parameter block has been received.
4. The new values of \$BPSAM (bytes per sample), \$NSAMS (number of samples in ring buffer), \$QTNUM (total quench sample count), and \$HTNUM (total heater sample count) are stored.
5. The new parameters are copied from \$RBUF (the receive buffer) to \$PBUF (the parameter block buffer).

3.8.4 START

This label marks the section which handles the actual start-up of scan data acquisition. First, FFE sets up the eight ADAC command words \$ADIAD, \$AD2AD, etc. (see SYSCM.MAC) and loads the ADAC device registers. To do this, it must separate the requested channels between the two distinct ADAC systems. It must decide which channels must be read by unit #1, which by unit #2, starting channel numbers for each, and so forth. The following algorithm is used to accomplish this:

1. Initialize the buffer address parameters \$AD1BA and \$AD2BA to the ring buffer bottom address, \$SBUFFR. Initialize the other six command words to zero.
2. Decrement the low channel number to put it on a 0-63 basis, rather than 1-64. Thus, ADAC unit #1 handles channels 0-31, and unit #2 handles channels 32-63.
3. If the low channel is less than or equal to 31, skip to step 4; otherwise, all of the channels are within the unit #2 range. Set \$AD2WC equal to the negated channel count. Subtract 32 from the low channel number to get the unit #2 low channel (for example, channel 32 is the first channel, or channel 0, on unit #2). Swap this to a high byte, and set the 1012EX command bits (see SYSDF.MAC). Move the result to \$AD2AD. Set the interrupt enable and go bits in \$AD2DM. Go to step 7.
4. Calculate the high channel number by adding the number of channels to the low channel. The low channel is equal to the unit #1 low channel, so swap it to a high byte, set the command bits, and move it to \$AD1AD. Set the \$AD1DM command bits.
5. If the high channel is greater than 32, skip to step 6; otherwise, all the channels are in the unit #1 range. Move the negated channel count to \$AD2WC and go to step 7.
6. At this point, it has been determined that both ADAC units are required to read the requested channels. Subtract 32 from the low channel number (which gives a negated channel count for unit #1) and load the result into \$AD1WC. The number of channels which are left is negated and moved to \$AD2WC. The number of channels for unit #1 is multiplied by 2, and the product added to \$AD2BA to offset the unit #2 buffer address from that of unit #1. Command bits are set in \$AD2DM and \$AD2AD (if both units are used, the unit #2 low channel is always 0).

7. At this point, the command words for the ADAC systems required have been set up correctly; those for unused systems are left at zero (except for the bus address command words, which are always nonzero). Both ADAC 1012EX units are now made ready by testing the data buffer registers.
8. The ADAC 1620DMA bus address, word count, and CSR registers are all loaded from the command words.
9. Both "ready" flags (GF.RD1 and GF.RD2) are set in the global flag word \$GFLAG. The flags are also set in \$RDMSK, but only if the appropriate ADAC system is being used. Thus, when \$RDMSK is used to clear the "ready" flags (see CLINT.MAC), only the flags for the active units are actually cleared.

The "end of transient" global flag (GF.EOT) is now cleared (whether or not it was set). The sample number \$SAMPL is initialized to zero. The appropriate interrupt enable bits are set in the DRV11 CSR, depending on which transients were enabled in the parameter block. FFE then begins the clock start-up procedure.

To start the clock, FFE begins by negating the scan rate parameter and loading it into the clock buffer/preset register, to establish the overflow count. Then, the clock control parameter is examined. If the clock control parameter specifies internal control, the clock frequency parameter is multiplied by 10(8), which results in the correct rate bits (bits 3-5) to control the clock. The operating mode is incremented to scan mode, and the initial clock interrupt entry is set to #SCLIN1 (see CLINT.MAC). If, however, the clock control parameter specifies external control, the clock frequency parameter is a dummy that is not used. Instead, the clock rate bits are set to the ST1 option, which specifies that the oscillating signal to be counted will be supplied from outside (in this case, from the PDP-11/60). The initial clock interrupt entry is set to #SCLIN0, but \$MODE is not changed until the first clock interrupt. After the clock rate bits are set, the "ST2 interrupt enable", "overflow interrupt enable", "mode 1", and "go" bits are OR'ED in. The clock is then started by loading the CSR with the assembled command word, and the subroutine returns with a success code in word 1 of \$XBUF.

3.8.5 \$RESTR

The subroutine \$RESTR is called to process host command 6. It causes FFE to begin scan data acquisition using the parameter block which currently exists in \$PBUF, and it is meant to be called only after a transient collection episode (i.e., after a transient is over and all the data have been retrieved by the host). The subroutine first checks to see that the system is in prescan mode, and that the "end-of-transient" global flag is set. These two conditions can occur simultaneously only after the end-of-transient data transmission (or when this is simulated via command 8). If either of these two conditions is not true, the subroutine returns with a "not ready" error in word 1 of \$XBUF; otherwise, the transient interrupt number in the parameter block is cleared and the routine branches to the START label (see above).

3.9 CLINT.MAC

This file contains the clock interrupt handler code. After one of the start-up routines (see START.MAC) has started the clock, it begins to interrupt at the overflow vector each time the clock count, as defined by the buffer/preset register, overflows to zero. The overflow vector normally points to \$CLINT, but there are two alternate entries as described below.

3.9.1 \$CLIN0

This is an alternate clock handler entry point, which is used only when the clock is being run under external control. In this case, the start-up routine (see START.MAC) does not put the system into

scanning mode, since it cannot tell how long it might take to get the first interrupt from the host. Instead, it changes the clock overflow vector to point at \$CLIN0. When the first interrupt occurs, \$CLIN0 increments \$MODE to scan mode, and replaces the clock vector entry with \$CLIN1 (see below).

3.9.2 \$CLIN1

This is an alternate clock handler entry point, which is used for the first few samples. If the scan rate is slow, it is possible that the host could ask for a scan average before enough samples have been collected. For example, if the scan rate was 30 seconds between samples, and the parameter block specified 10 samples in an average, it would take 5 minutes to collect enough samples to calculate an average. \$CLIN1 is meant to notify the system when enough data is available for averaging. At each clock interrupt, the sample number \$SAMPL is compared to the "samples per average" parameter in the parameter block. When they become equal, "the scan data available" global flag (GF.SDA) is set and the clock vector is changed to \$CLINT for all subsequent interrupts.

3.9.3 \$CLINT

This symbol is the main clock interrupt entry. Immediately upon entry, \$CLINT loads both ADAC 1012EX CSR's with their respective command words. This starts both A/D conversion processes; if one of the ADAC units is not being used, its command word is zero, and loading the CSR with zero has no effect. \$CLINT then checks both of the global "ready" flags (GF.RD1 and GF.RD2) to see if both ADAC systems had completed their previous samples. If either flag is not set, then a new sample was started before one of the previous ones was finished, so a fatal error is declared. If both units are ready, the active "ready" flags are cleared using \$RDMSK (see SYSCM.MAC). \$MODE is then tested to determine the system state. If negative, the system is still in normal scan mode. If zero, it is in primary posttrigger (PT1) mode, and if positive it is in secondary posttrigger (PT2) mode.

3.9.4 SCAN MODE

In scan mode, \$CLINT next tests the scan average access flag (GF.ACC). If clear, it branches to the "next sample" calculation (see below). If set, a scan average request is being processed, and FFE is accessing data in the ring buffer to calculate the averages. In this case, \$CLINT compares the current sample number (\$SAMPL) to the sample number where the averaging routine is currently reading data (\$AVSAM). If they are equal, the scan data being averaged have been overrun by new results, and the average overrun flag (GF.AVO) is set to inform the scan average routine of this fact. After all of these checks, \$CLINT branches to the "next sample" routine below.

3.9.5 PT1 Mode

In PT1 mode, \$CLINT decrements the number of PT1 samples (\$P1NUM). If the result is nonzero, the routine branches to the "next sample" routine below. If zero, the clock buffer/preset register is loaded with the negated PT2 clock rate parameter, which has been stored in \$P2RAT. \$MODE is incremented, so that upon the next clock interrupt the system will be in PT2 mode. \$CLINT then branches to the "next sample" routine.

3.9.6 PT2 Mode

In PT2 mode, \$CLINT decrements the number of PT2 samples (\$P2NUM). If the result is nonzero, the routine branches to the "next sample" routine below. If zero, transient collection is finished and the clock is stopped by clearing the KVV11-A CSR. The transient sample number \$STRSAM (the value of \$SAMPL when the transient interrupt occurred) is retrieved, then the

requested number of pretrigger samples is subtracted from it to obtain a sample number representing the beginning of the transient data (note that if the sample number goes below zero in the subtraction, it is "wrapped around" to the top of the ring buffer by adding \$NSAMS to it). The host system is interrupted by setting, and then clearing, the appropriate bit in the DRV12 output buffer. Finally, the "end of transient" flag (GF.EOT) is set, \$MODE is incremented to "posttransient" mode, and the interrupt routine exits.

3.9.7 NEXT Sample

The next sample number is calculated simply by incrementing the current sample number, \$SAMPL. The ADAC command words are then adjusted, according to the new value of \$SAMPL. If the new \$SAMPL is less than the total sample count \$NSAMS, then \$AD1BA, the bus address for ADAC unit #1, is extended by \$BPSAM, the number of bytes per sample. If the result of this addition overflows a 16-bit word, then the extended address bits in \$AD1DM, the 1620DMA CSR command, are incremented; otherwise, this word is not touched. A similar procedure is then followed for ADAC unit #2. These actions have the effect of advancing the ring buffer address for the next sample.

If, on the other hand, the resultant \$SAMPL is equal to \$NSAMS, this means that the top of the ring buffer has been reached. In this case, \$SAMPL is reset to zero. The ADAC bus address commands, \$AD1BA and \$AD2BA, are reset to their base values, \$BUPFR and \$A2BAS, respectively. Finally, the extended address bits in both 1620DMA CSR words, \$AD1DM and \$AD2DM, are cleared. These actions set both the sample number and the physical DMA addresses back to the bottom of the ring buffer.

3.9.8 Clock Reenable

As the last action taken by the clock interrupt handler during data collection, the clock itself is reenabled. First, the KWV11-A flag overrun bit is checked. If this bit is set, it means that the clock overflowed (tried to interrupt) again while the present interrupt was being processed; in other words, the clock is running too fast for the program to keep up with. In this case, a fatal system error is declared. If the overrun bit is not set, the clock overflow bit (bit 7) is cleared, which effectively reenables the clock for the next interrupt. The interrupt handler then exits.

The programmer should note that all of this handler code is executed at each clock interrupt, at up to 1000 times per second. Therefore, every effort has been made to keep this module as small and as fast as possible. The programmer should keep this in mind if any modifications are made to CLINT.MAC.

3.10 STINT.MAC

This file contains \$STINT, the interrupt handler for the KWV11-A ST2 interrupt vector. While the ST2 interrupt is not currently used by FFE, it is necessary to keep the interrupt enabled. If ST2 fires due to some unexpected external pulse or electrical noise, the ST2 flag (bit 15) will be set in the clock CSR. If it then fires again while the ST2 flag is set, the flag overrun bit (bit 12) is asserted. This is detected by FFE as a fatal error condition (see CLINT.MAC). To prevent this from happening, \$STINT clears the ST2 bit whenever any ST2 interrupts occur. This is the only function of this module.

3.11 ADINT.MAC

This file contains the interrupt handlers for the ADAC 1620DMA devices. There are two labels, \$D1INT and \$D2INT, one for each ADAC unit, and the two interrupts are handled exactly the same. The appropriate routine is entered each time a sample (i.e., a sequential block of DMA channels) has been read, and the "ready" bit in the 1620DMA CSR is asserted. The handling procedure is as follows:

1. Test the error bit in the CSR. If it is set, declare a fatal error and exit; otherwise, continue to step 2.
2. Test the corresponding ADAC 1012EX data buffer register to clear the 1012EX ready bit.
3. Load the 1620DMA word count, bus address, and CSR registers from the corresponding command words, which have been prepared in advance by \$CLINT (see CLINT.MAC). This prepares the ADAC system to collect the next sample, all except for the 1012EX CSR command which actually initiates conversion (this initiation takes place at the next clock interrupt).
4. Set the appropriate "ready" flag in the global flag word, to notify \$CLINT that the sample has completed.
5. Exit the interrupt.

3.12 TRINT.MAC

This file contains \$TRINT, the interrupt handler for the DRV11 quench and heater pulse interrupts. It is entered upon the occurrence of either one of these transients. The two interrupts have different vectors, but they point to a common address. They are distinguished by the handler via the C-bit which is set in the PSW for the heater pulse vector, but not set for the quench vector. The handling of the interrupts is as follows:

1. Test \$MODE. If not in scan mode, exit immediately. This is to prevent an interrupt from being processed when the clock is not running.
2. Determine the type of interrupt by testing the C-bit in the processor status word.
3. Clear the DRV11 interrupt enable bit for the interrupting transient to prevent it from interrupting again.
4. Move the appropriate interrupt identifier to the parameter block, so the host can determine which transient occurred.
5. Negate the posttrigger #1 clock rate for the transient and load it into the clock buffer/preset register. This means that at the next clock overflow, the clock will start using this new counter value.
6. Negate the posttrigger #2 clock rate and store it in \$P2RAT for the clock handler to access later (see CLINT.MAC).
7. Store the posttrigger sample counts in \$P1NUM and \$P2NUM.
8. Increment \$MODE to "primary posttrigger" status.
9. Store the current sample number \$SAMPL in \$STRSAM.
10. Exit the interrupt.

3.13 XPARB.MAC

This file contains \$XPARB, the command table routine called to transmit the current parameter block back to the host system. It is called by \$IDLER to process host command 2. First, the routine checks the "parameter block received" flag (GF.PAR) to see if at least one parameter block has been received yet. If not, a "not ready" error is returned. Otherwise, the routine loads a success code into word 1 of \$XBUF, and copies \$PBUF (containing the parameter block) right after it. The subroutine then returns to \$IDLER, which transmits the data in \$XBUF back to the host.

3.14 XSCAN.MAC

This file contains \$XSCAN, the command table routine called to compute scan averages and transmit them to the host. It is called by \$IDLER to process host command 5. This command is successful only during scan data acquisition. The procedure followed by \$XSCAN is as follows:

1. Test the "scan data available" flag (GF.SDA) to see if scanning has started, and if enough samples have been collected to satisfy the "samples per average" parameter (see CLINT.MAC). If not, a "not ready" error code is loaded into word 1 of \$XBUF and the routine returns.
2. Test \$MODE to see if the system is still in scan mode. If not, an error code indicating that the system is collecting transient data is loaded into word 1 of \$XBUF, and the subroutine returns.
3. Lock out interrupts by setting the processor priority to 7. Get the current sample number \$SAMPL, and subtract the "samples per average" parameter from it. Decrement the result one more time since the current sample is not finished yet. The result is the sample number where averaging will start. Store it in \$AVSAM (see SYSCM.MAC) and set the "averaging access" flag (GF.ACC) to notify \$CLINT that averaging is in progress (see CLINT.MAC). Return the processor to priority zero.
4. Clear 128 words of the transmit buffer \$XBUF, starting at \$XBUF+2, to use in double precision additions on a possible 64 channels.
5. Convert the sample number \$AVSAM to a memory management bias and displacement by calling subroutine \$S2ADR (see S2ADR.MAC). After this call, R1 contains the virtual address of the first piece of data, and KPAR6 of the MMU has been loaded with the correct relocation bias. All data points within a particular sample can be accessed using R1 without changing KPAR6.
6. Using autoincrement mode on R1, move the next data point to the stack and bump R1 to the next point. Test bit 11 (the MSB for the 12-bit data) in the stacked result. If not set, the data point is 12-bit positive, so add it in double-precision fashion to the next two words in \$XBUF. If bit 11 is set, the point is 12-bit negative. In this case, extend the sign by OR'ing in 170000(8) before the double-precision addition, and add 177777(8) to the high word of the sum after the double-precision addition.
7. When all channels within the sample have been read, increment \$AVSAM to the next sample, and return to step 5. Do this a number of times equal to the "samples per average".
8. When all the data have been accessed and summed, clear the GF.ACC flag. Test \$MODE again to make sure a transient has not occurred since the subroutine was entered. If it has, return with the transient notification code in word 1 of \$XBUF.
9. Check the flag GF.AVO to see if an average overrun occurred (see CLINT.MAC). If so, load a "data overrun" error code into word 1 of \$XBUF, and clear the flag.
10. If no errors have occurred, load the success code into word 1 of \$XBUF.
11. Begin dividing to calculate the averages. Move the double-precision sums in \$XBUF to R0 and R1, and then divide by the "samples per average" parameter. If the PSW V-bit is set after any of these divides, it means that the division was not successful, so load a "bad divide" error code into word 1 of \$XBUF.
12. Move the result back into \$XBUF. Note that any negative averages have an extended sign at this point. This makes the 12-bit negative number look like a standard negative integer to a FORTRAN host program.
13. Repeat steps 11 and 12 until all active channels have been done. Return to \$IDLER, which will transmit the final results back to the host.

3.15 \$S2ADR.MAC

This file contains \$S2ADR, a utility subroutine used to convert a system sample number into a memory management relocation bias and displacement. It is called using R5, so that the address of the sample number can be passed as an argument.

The sample number is retrieved and multiplied by \$BPSAM, the number of bytes per sample. The base address of the ring buffer, \$SBUFR, is then added to the product. This results in a double-precision physical address. This address is divided by 100(8) to produce the MMU relocation bias (as the quotient), and the displacement (as the remainder). The bias is loaded into MMU register KPAR6. The displacement is OR'ED with 140000(8) to specify a page 6 virtual address, and this virtual address is returned to the caller in register R1.

3.16 XTRAN.MAC

This file contains command table subroutines related to transient data handling. None of these routines is successful unless the system is at an end-of-transient condition.

3.16.1 \$XTRN0

This subroutine is called to process host command 3, and it is used to initiate transient data transmission to the host. The procedure used is as follows:

1. Test the "end of transient" flag (GF.EOT). If not set, return with a "not ready" error in word 1 of \$XBUF.
2. If the system is in "prescan" mode instead of "end-of-transient" mode, it means that the host has asked for the transient data again after it has all been sent once. This is permitted, so \$XTRN0 sets \$MODE back to +2 for the duration of data transmission.
3. Clear the local channel and block variables, and set up a channel counter from the channel count parameter.
4. Retrieve the total transient sample count for the appropriate transient interrupt and store it locally.
5. Retrieve the starting sample number of the transient data, \$STRSAM, and store it in \$SAMPL (\$SAMPL is no longer needed for its normal use, since all data collection has stopped).
6. Branch to XFER (see below).

3.16.2 \$XTRN

This subroutine is called to process host command 4, and is used to transmit all blocks of transient data subsequent to the first one. It is called repeatedly by the host for sequential blocks of data, until the subroutine returns with an "end-of-file" error code in word 1 of \$XBUF. The subroutine returns a "not ready" error if the system is not in "end-of-transient" mode (mode 2), or if the current block number is zero (indicating that the initializing routine \$XTRN0 has not been called yet). Otherwise it skips directly to XFER (see below).

3.16.3 XFER

This section of common code does the actual transfer of results to \$XBUF for transmission. The procedure used is as follows:

1. Increment the block number and move it to word 1 of \$XBUF. \$XTRN0 should return block number 1, and subsequent blocks should have block numbers 2, 3, etc., until the last block which contains an "end-of-file" error code instead of a block number. Note that the "end-of-file" block will still contain some data.
2. Convert the current sample number to a memory management relocation bias and a virtual address by calling \$S2ADR (see S2ADR.MAC). The virtual address of the sample is returned in R1.
3. Add a channel offset to R1 to produce the virtual address of the channel of interest within the sample block. Get the result at this address and test bit 11. If it is set, set bits 12-15 to extend the sign of the negative result. Move this result to the next slot in \$XBUF.
4. Decrement the transient sample count. If the result is zero, go to step 5; otherwise increment the sample number \$SAMPL and go to step 6.
5. When the sample count goes to zero, all the data for the current channel have been retrieved. Increase the channel offset by 2, and compare the result with the channel offset limit. If they are equal, go to DONE (see below); otherwise set up the sample count again and set \$SAMPL back to \$TRSAM.
6. If \$XBUF is full, return to the caller; otherwise loop back to step 2.

3.16.4 \$KILTR

This subroutine is called to process host command 8. It is used to effectively discard transient data if the host wishes to restart scanning without taking the time to retrieve all of the transient results. It simulates the "end-of-file" condition which normally occurs at the end-of-transient data transmission. It returns a "not ready" error if called when \$MODE is not equal to 2; otherwise it skips to DONE (see below).

3.16.5 DONE

At this point, transient data transmission is finished (or has been killed via \$KILTR). The "end-of-file" error code is moved to word 1 of \$XBUF, replacing the block number which was there. The system mode is set back to -2 (prescan mode). The local block number variable is cleared to avoid erroneous calls to \$XTRN, and the subroutine returns.

3.17 RBOOT.MAC

This file contains \$RBOOT, which is called to process host command 7. Its purpose is to reboot the LSI-11/23 so that a new downline load operation can take place. It clears the BDV-11 page control register, terminates the link by calling FETERM, performs a RESET instruction to initialize the system, and jumps to the system bootstrap address. Obviously, the subroutine never executes a return, nor does FFE transmit any reply to the host for this command.

3.18 ERROR.MAC

This file contains system error handling routines. These routines handle the front-end system errors, as opposed to errors in processing host commands, which are handled in the command subroutines.

3.18.1 \$FATAL

This routine handles fatal system errors. When a fatal error is detected, it is assumed that the system can no longer reliably perform its functions. This routine stops all activity on the system except for the communications link. The link is used to inform the host of the fatal error condition and to transmit some diagnostic data which should be of use in determining the problem. The system will ignore all commands from the host until it gets a reboot command to start things over again.

\$FATAL is entered under two different conditions:

1. FFE declares a fatal error explicitly via an EMT instruction, or
2. an unexpected error causes a hardware trap to one of the system trap vectors.

LOWCR.MAC lists all of the trap vectors which use \$FATAL. The unexpected traps are distinguished from each other (and from the EMT trap) by different condition code bits set in the trap vector PSW's. The sequence of events in \$FATAL is as follows:

1. Save the PSW on the stack to preserve the condition code bits at entry.
2. Test the "fatal error" flag (GF.FTL). If set, \$FATAL has already been entered once, so exit immediately. If clear, set the flag.
3. Save registers R5-R0 on the stack, and retrieve the saved PSW from it. Replace the buffer address in the transmit argument block \$XARG by \$EBUF, the error buffer address. Thus, any further transmissions to the host will be from \$EBUF, regardless of what may happen to \$XBUF.
4. If the entry PSW had any condition code bits set, the error was not an EMT trap. In this case, put an "illegal instruction" error into the error buffer. If the PSW bits show that the error was from the MMU, load the MMU status registers into the buffer. Go to step 7.
5. If the error was an EMT, retrieve the trapped PC from the stack and locate the EMT instruction. Add the low byte to the current PC so that a branch to the correct handling routine is made, where appropriate error codes are loaded into the buffer.
6. Disable the clock and the DRV11. Interrupt the host system to inform it of the error condition.
7. Load the current stack size into the error buffer, and then copy the entire system stack into it.
8. Restore registers R5-R0 and exit.

After \$FATAL exits, any messages sent to the host will have the following format:

- Word 1 -- IE.FHE (fatal error code)
- Word 2 -- IE.XXX (error type subcode)
 - Error-dependent data--number of words varies, can be zero
 - -1 (to mark end of error-dependent data)
 - Number of words in the stack

4. PROGRAM MAINTENANCE

This chapter outlines procedures for maintaining FFE. It lists some recommendations for editing procedures, and describes how to assemble, link, and debug the program.

4.1 Files

All files concerned exclusively with FFE are located in account [160,160] on the host PDP-11/60. Although the MACRO-11 source files were described in Chapter 3, the entire file set is listed below for completeness.

FFE FILE SET

ADINT.MAC	ADAC 1620DMA interrupt handlers
CLINT.MAC	Clock interrupt handler
ERROR.MAC	Fatal error handler
FESUBS.MAC	Link subroutines
IDLER.MAC	Task link/idle loop
INITL.MAC	Task initialization routines
LOWCR.MAC	Low memory layout
RBOOT.MAC	Reboot routine
S2ADR.MAC	Sample to address conversion routine
START.MAC	Scanning startup routines
STINT.MAC	Clock ST2 interrupt handler
SYSCM.MAC	System common data storage area
SYSDF.MAC	System definitions
TRINT.MAC	Transient interrupt handler
WTLNK.MAC	Link completion wait routine
XPARB.MAC	Transmit parameter block routine
XSCAN.MAC	Transmit scan averages routine
XTRAN.MAC	Transmit transient data routine
FFETST.FTN	FFE test program
COPY.CMD	FFE.TSK copy command file
FFEASM.CMD	FFE assembly command file
FFETKB.CMD	FFE taskbuild command file
FFETST.CMD	FFETST taskbuild command file
FFE.TSK	FFE task image
FFETST.TSK	FFETST task image
LSILDR.TSK	Required to run DLN from account
FFE.MAP	FFE taskbuilder map

4.2 Editing Existing Files

If the need arises to edit some of the MACRO-11 source files, the programmer should maintain the file conventions listed in Chapter 3. In addition, the following points should be noted:

1. **SYSDF.MAC** can be modified for changed device addresses, new devices, or changes in buffer sizes. New parameters can be added to the parameter block by adding new symbolic offsets onto the end of the present list; however, the existing parameters may not be rearranged without extensive editing of other parts of the program. New global flags can be added, up to a total of 16.
2. New global variables may easily be added to **SYSCM.MAC** when needed (but beware of rearranging the existing ones). New command codes can be added to the program by adding new entries at the bottom of the **SCMDTB** list, right before the definition of **KMAX**. The entries must be the names of subroutines, which will presumably be added to existing files or created in new files.
3. There are no "safe" registers in the program. All registers in the set **R0-R5** are presumed volatile throughout the program, except in those subroutines explicitly stating their subroutine usage (e.g., **SS2ADR**). The **FESUBS** link routines preserve **R4** and **R5**. Interrupt handling routines, of course, must preserve all registers.
4. The interrupt routines all run at processor priority 7 and are meant to run as fast as possible. Beware of adding excessive code to the interrupt handlers.
5. New fatal error conditions are easily added to the program with new **EMT** instructions. The easiest way of finding out how this works is by examining the **\$FATAL** routine in **ERROR.MAC**.
6. Much of the program structure is based on the assumption of the two **ADAC** systems, with 32 input channels per system. If this situation changes, the program will need extensive modification.
7. Interrupt vectors for new devices can easily be added to **LOWCR.MAC**.

4.3 Creating New Files

It may become necessary to create new source files to be incorporated into **FFE**. In this case, the programmer is mostly on his own, as long as the new code is compatible with the old. It is recommended that the file conventions described in Chapter 3 be followed in creating any new source files. In addition, adding new source files means that **FFEASM.CMD** and **FFETKB.CMD** must be edited (see below).

4.4 Source File Assembly

To assemble all of the **MACRO-11** source files, use the indirect command file **FFEASM.CMD**. It is an **MCR** command file, so it is invoked by typing:

```
>@FFEASM
```

The command file asks whether or not source listings are desired. If the answer is yes, source listings are created and spooled, and object files are created. If not, object files only are created.

If the object files already exist and only one or two files need reassembly, they can be done by hand. The command line should be of the form

```
>MAC NAME=SYSDF,NAME
```

where "NAME" is the file being reassembled. If **SYSDF.MAC** is changed, all of the files should be reassembled.

If any new source files are created for insertion into **FFE**, this command file should be edited to add the name of the new file. A **MACRO-11** assembly command similar to all of the others must be inserted into **FFEASM.CMD** twice; once into the section which creates listings, and once into the "no listings" section.

- System stack from top down, ending with the following items:

Trap PSW
Trap PC
R5
R4
R3
R2
R1
R0

- -1's to fill out \$EBUF

Word 33 -- Start of global common variables in SYSCM.MAC

3.18.2 \$NONSI

This is a label on a RTI instruction, which is used as the nonsense interrupt entry. An interrupt occurring at any vector which points at this address will simply return immediately.

3.18.3 \$LKERR

This routine handles link errors. It is entered via an IOT trap instruction. First, \$LKERR calls FETERM to kill the link. If a soft error occurred (e.g., bad checksum), the link error variable LNKERR will be clear after this call. In this case, \$LKERR calls FEINIT to reinitialize the link, and then it exits. If a hardware problem exists, LNKERR will be nonzero. In this case a HALT is executed to stop the processor.

4.5 Taskbuilding

FFE can easily be taskbuilt by using the command file FFETKB.CMD, as follows:

```
>TKB @FFETKB
```

A task file and a map file are always generated, although the map file is not automatically spooled. The map file is essential for debugging.

If new source files are to be inserted into FFE, this command file must be edited. The input modules are listed, one per line, in the file. LOWCR must remain as the first module in the list, and INITL must remain as the last module in the list. Otherwise, it does not matter where in the list the new modules are inserted.

4.6 Testing and Debugging

A FORTRAN test program called FFETST is available for testing FFE. After FFE.TSK is loaded into the front end using DLN, run FFETST. The program first asks "WHICH XB: UNIT: "; enter the number of the XB: unit to which you downline loaded FFE. FFETST then begins asking for commands. Any of the legal command values may be entered. Each time a command is entered, it is sent to the front end, and the reply is printed on the terminal as a block of octal numbers. If command 1 is entered, the program asks for a list of parameters to send over. If command 7 (reboot) is entered, the program exits since no reply is expected, and it can no longer communicate with the front end until a new downline load has been done.

If there are bugs in FFE, it can be difficult to find out where they are. If a system fatal error occurs, the system status is fairly well summarized in the data returned from \$EBUF. If the system is just halting, or if other information on the task is needed, it may be necessary to connect a terminal to the front end DLV11 interface. A console terminal allows the use of console ODT to access any location in memory when the system halts, or when the programmer interrupts FFE by hitting the BREAK key on the terminal. It is even possible to resume FFE where it was interrupted using the P (Proceed) command of ODT, but it does not always work due to link timeouts and other considerations.

In either case (fatal error return or console ODT), use FFE.MAP to extract as much system status information as possible. Then, try to trace down exactly what the system was doing at the time of the error. If no error can be found, do not forget the possibility of hardware problems that are undetectable by the software.

ORNL/CSD/TM-198

INTERNAL DISTRIBUTION

1. R. K. Abercrombie
2. L. L. Anthony
3. R. W. Browe,¹
4. R. D. Burris
5. H. P. Carter/G. E. Whitesides/X-10 CSD Library
- 6-7. Central Research Library
8. J. G. Craven
9. Document Reference Section
10. S. S. Gould
11. D. E. Greenwood
12. K. P. Hearn
13. J. M. Jansen
14. K-25 CSD Library
15. K-25 Plant Library
- 16-18. Laboratory Records
19. Laboratory Records — RC
- 20-24. L. R. Layman
25. S. H. Merriman
- 26-30. D. L. Million
31. ORNL Patent Office
32. S. S. Stevens
33. P. L. Walstrom

EXTERNAL DISTRIBUTION

34. Division of Engineering, Mathematics and Geosciences, DOE, Washington, D.C. 20545
35. Office of Assistant Manager for Energy Research and Development, DOE/ORO, Oak Ridge, TN 37830
36. J. N. Rogers, Division #324, Sandia Laboratories, Livermore, CA 94550
- 37-63. Technical Information Center, P. O. Box 62, Oak Ridge, TN 37830