SINGLE PASS COLLIDER MEMO

CN-

SLAC-CN--249

DE84 011773

AUTHOR: Sung Joo Yoo

DATE: 9/26/83

REPLACES CN#

TITLE: PHASE & AMPLITUDE CONTROL SYSTEM FOR STANFORD LINEAR ACCELERATOR

ABSTRACT

During the summer in 1983, I worked on the computer controlled phase and amplitude detection system to measure and stabilize the RF pulses in the two mile Stanford Linear Accelerator. This system measures the instantaneous phase and amplitude of a 1 micro-second 2856 MHz RF pulse at a 180 Hz rate. This will be used for phase feedback control, and also for phase and amplitude jitter measurement. The program, which was originally written by John Cox and Keith Jobe, has been modified to improve the function of the system. The software algorithms used in the measurement are described, as is the performance of the prototype phase and amplitude detector system.

MASTER

INTRODUCTION

This paper describes an RF phase and amplitude measurement system being developed for the Stanford Linear Collider (SLC) Project. The system is broken into four major parts: a computer control system, a timing system, a remote controller, and a remote RF and Analog heads (Figure 1).

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.

NOTICE
PORTIONS OF THIS REPORT ARE ILLEGIBLE. It has been reproduced from the best available copy to permit the broadest possible availability.
DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

EWB

Figure 1: System block diagram

<Control System>

The SLAC accelerator is being instrumented on a new control system which is based on VAX main computer with a large number (~100) of remote computing nodes. Each node features an Intel 8086/8087 micro computer connected to the host by a broad band CATV communication system (SLCNET). The remote processors (cluster micros) control all

devices in a geographically designed area. The cluster micro communicates with all peripherals through a new serial CAMAC channel supporting up to 16 crates. The high speed control of klystrons is done by a Parallel Input Output Module (PIOP) resident in CAMAC.

<Timing System>

The timing of all accelerator systems is derived from a new timing system based on a Programmable Delay Unit (PDU). The PDU is a 16 channel digital delay unit that operates in the CAMAC environment. The Accelerator's 476 MHz main drive line is the frequency source for all timing delays. The PDU detects the fiducial from the main drive and counts cycles of a 119 MHz subharmonic of the main drive to generate timely triggers on a special CAMAC upper backplane bus.

<Remote Controller>

The PIOP is an 8088 based controller designed for the high speed control and monitoring of the pulse accelerator klystrons at SLAC. The processor is interrupt driven, and monitors all signals of interest every 2.7 milli-seconds(at 360 Hz). Detailed discussions of PIOP can be found in reference 1.

<Remote Head>

The remote heads for use with the PIOP contain all the analog and RF circuitry. The timing for any sampling is derived from the PIOP via the differential cable. The firmware in the PIOP will, upon receiving an interrupt, read the head and assert any new values that

heads. The pulse measurement uses a double balanced mixer as a nulling detector, and a voltage variable phase shifter in the signal path which is adjusted to null the mixer. If the null is achieved, the nulling detector and voltage variable phase shifter can be used in a feedback system to maintain the original phase relationship between the signal and reference lines. Additionally, if the phase vs. voltage characteristic (Fig 2) is known, an absolute measurement of the phase difference between signal and reference can be determined.

Figure 2: Phase Shifter

hence $\phi_s - \phi_r = 0$ or π

By knowing the sign of the slope, we can determine whether $\phi_s - \phi_r = 0$ or π .

This zero impolation is done in a try and error manner in the LOCKER routine. First it calculates $V_{mix} = V_{out1} + \pi/2 - V_{out1} - \pi/2$ at zero phase shift. And then it calculates V_{mix} at phase shift = 90 (please see flow chart for details). Next it checks if V_{mix} has changed its sign or not. If it did, it sets phase shift = 0 again. Similarly it adds 90°, 45°, 22.5°, 11.3°, 5.6°, 2.8° . . . to the phase shift, and subtracts the same amount each time V_{mix} changes its sign. Repeating this 8 times, we can approach the crossing within 0.7 degrees ($180/2^8$) accuracy.

The phase shift voltage is changed by changing the content of 8 bit memory at 4004H [DAC].

STEP 3: Once phase shift voltage is locked to the neighbor of zero crossing, more accurate zero crossing can be measured by Newton-Rapson Algorithm

It reads V_{mix} at two DAC's separated by small amount¹ and calculates V_{zero} from² these two pairs of coordinates:

$$V_{zero} = DAC(J) - \frac{(DAC(J+1) - DAC(J)) * V_{mix}(DAC(J))}{(V_{mix}(DAC(J+1)) - V_{mix}(DAC(J)))}$$

¹ typically 0.7 degrees

² Cosine function is essentially straight line around zero crossing

hence $\phi_s - \phi_r = 0$ or π

By knowing the sign of the slope, we can determine whether $\phi_s - \phi_r = 0$ or π .

This zero impolation is done in a try and error manner in the LOCKER routine. First it calculates $V_{mix} = V_{out1} + \pi/2 - V_{out1} - \pi/2$ at zero phase shift. And then it calculates V_{mix} at phase shift = 90 (please see flow chart for details). Next it checks if V_{mix} has changed its sign or not. If it did, it sets phase shift = 0 again. Similarly it adds 90°, 45°, 22.5°, 11.3°, 5.6°, 2.8° . . . to the phase shift, and subtracts the same amount each time V_{mix} changes its sign. Repeating this 8 times, we can approach the crossing within 0.7 degrees ($180/2^8$) accuracy.

The phase shift voltage is changed by changing the content of 8 bit memory at 4004H [DAC].

STEP 3: Once phase shift voltage is locked to the neighbor of zero crossing, more accurate zero crossing can be measured by Newton-Rapson Algorithm

It reads V_{mix} at two DAC's separated by small amount¹ and calculates V_{zero} from² these two pairs of coordinates:

$$V_{zero} = DAC(J) - \frac{(DAC(J+1) - DAC(J)) * V_{mix}(DAC(J))}{(V_{mix}(DAC(J+1)) - V_{mix}(DAC(J)))}$$

¹ typically 0.7 degrees

² Cosine function is essentially straight line around zero crossing

Figure 3: Zero Crossing Approximation in LOCKER routine.

In the case zero-crossing occurring around the edge of the phase range (0 - 180 degrees), it checks whether shift of DAC will cause overflow or not. If it does, it moves sampling DAC's by another small amount, and does V_{zero} calculation in the same way (figure 5). Phase vs Voltage characteristic of the shifter is then used to convert the V_{zero} to phase measurement. If slope < 0, 180 degree offset is added to the measured phase.

Finally it checks if the calculated V_{zero} is close to the zero crossing approximated in STEP 2. If the phase loses its lock, LOCKER routine will be reexecuted to regain the lock of the phase. Thus

Figure 4: Zero Impolation in TRACKER routine **Figure 5:** Shift of Measure Points at the Edge

routine will be reexecuted to regain the lock of the phase. Thus

est value near Vzero and tracks time varying phases.

STEP 4: From the measured voltages and calculated phases, we can get the averages and errors of these quantities. Average value after n measurements is calculated as;

$$\langle AVE \rangle_n = (\langle AVE \rangle_{n-1} + \text{Measurement}) / 2$$

Similarly Jitter is calculated as;

$$\text{JITTER}(n) = (\text{Measurement} - \langle AVE \rangle_n)$$

Note that the average value calculated in this manner deviates from the arithmetic average value. Because of this, calculated jitter is smaller than the actual jitter.

MEMORY AND ADDRESS SPACE

The memory space of the internal 8088 microprocessor resides entirely in the lower 64k of the 1 megabyte address space available. 8088 memory available from 00000 ~ FFFFF

1. ROM

The ROM is located between FFFF and C000 (or FFFF0 and C0000). The ROM is one or two 2732A's or one or two 2764's EPROM's. For the current PIOP, one 2764, at 8 k of memory is being considered.

2. RAM

The RAM is located between 0000 and 3FFF. The RAM is one or two 6116's, or one or two 6264's static memories. For the current PIOP, two 6116's at 4 k of memory is considered.

Figure 6: Memory Map

3. SHARED RAM

The shared RAM is a 6116 2K static memory device located between 5000 and 57FF. Access to this memory is shared between the DMA controller used in CAMAC block requests and 8088 resident processor. STATUS_BLK_1 and STATUS_BLK_2 is mapped in this location as following³

ADDRESS	NAME
5000	STATUS_BLK_1_LOC(0)
5002	STATUS_BLK_1_LOC(1)
.	.
.	.
503E	STATUS_BLK_1_LOC(1F)
5040	STATUS_BLK_2_LOC(0)
5042	STATUS_BLK_2_LOC(1)
.	.
.	.
507E	STATUS_BLK_2_LOC(1F)

³ addresses and indices are in hexadecimal

4. DATA LINK

The data link is mapped between 4000 and 40FF. The data is double buffered, and the processor is always stopped to allow an acknowledge from the remote device.

(MEMORY MAP):

ADDRESS	IDENTIFIERS		
4000	HEAD_LOC(0)	HEAD\$TYPE	LINK_ID_LOC HEAD(0)
4001	HEAD_LOC(1)	HEAD\$ID, HEAD_ECHO_LOC	
4002	HEAD_LOC(2)	COMMAND\$REG	HEAD(1)
4003	HEAD_LOC(3)	STATUS\$REG	
4004	HEAD_LOC(4)	DAC	HEAD(2)
4005	HEAD_LOC(5)	CLOCK\$CONTROL\$DIVISOR	
4006	HEAD_LOC(6)	WOBBLE\$CONTROL\$DIVISOR	HEAD(3)
4007	HEAD_LOC(7)		
.	.	.	.
.	.	.	.
.	.	.	.
400C	HEAD_LOC(C)	HEAD_FTP_DELAY_LOC	HEAD(6)
		HEAD_ADC_CHAN_LOC	
		HEAD_FTP_CHAN_LOC	
		HEAD_DAC_LOC	
		HEAD_ADC_LOC	
.	.	.	.
.	.	.	.
400F	HEAD_LOC(F)		
4010	HEAD_LOC(10)	WORD0	HEAD(8)

4011	HEAD_LOC(11)		
4012	HEAD_LOC(12)	WORD1	HEAD(9)
4013	HEAD_LOC(13)		
.	.	.	.
.	.	.	.
401E	HEAD_LOC(1E)	WORD7	HEAD(F)
401F	HEAD_LOC(1F)		
.	.	.	.
.	.	.	.
.	.	.	.
4048	HEAD_LOC(48)	HEAD_WATER_LOC	HEAD(24)
.	.	.	.
.	.	.	.
404D	HEAD_LOC(4D)	HEAD_DRIVE_LOC	
404E	HEAD_LOC(4E)		HEAD(27)
404F	HEAD_LOC(4F)	HEAD_STEP_MOTOR	.
.	.	.	.
.	.	.	.
40FE	HEAD_LOC(FE)		HEAD(7F)
40FF	HEAD_LOC(FF)	HEAD_RESET_LOC	

Simplified memory map of DATA LINK is shown in Table 1.

• A brief description of each register is as follows:

COMMAND REGISTER:

The command register is an eight bit R/W register which controls the operation of the digital system. Its structure is shown in Table 2.

Table 1: Memory Mapping of DATA LINK

RELATIVE ADDRESS	FUNCTION	COMMENT
0	TYPE NO.	read only
1	ID No.	read only
2	command register	
3	status register	read only
4	phase shift DAC	
5	clock control divisor	
6	wobble control divisor	
7-F	Available memory	
10-1F	ADC DATA BLOCK	
20-2F	available memory	
30-3F	offset ADC data block	
40-7F	available memory	
FF	special reset location	R/W causes reset

Table 2: Bit Mapping of COMMAND REGISTER

bit 0 Set + $\pi/2$ wobbler (negative TRUE)
 bit 1 Set - $\pi/2$ wobbler (negative TRUE)
 bit 2 spare
 bit 3 ADC data block offset
 bit 4 Head Interrupt enable
 bit 5 GREEN LED indicator (negative TRUE)
 bit 6 RED LED indicator (negative TRUE)
 bit 7 Asynchronous head digitize command

Status Register:

Status Register is 8 bit read only register structured as follows;

bit 0-5 spare

bit 6 wobble state 1 represents $+\pi/2$, 0 represents $-\pi/2$

bit 7 fresh update set by completion of ADC cycle

reset by a read from the status

register

Phase Shift DAC:

An 8 bit r/w register which reflects the setting of the phase shift control voltage. It is coded in straight binary, and the range of the DAC is 0-9.96 volts (00 represents 0 volts, FF 9.96 volts)

Clock Control Divisor:

An 8 bit R/W register which contains the number of 14.875 MHz clock cycles between the modulator-trigger pulse from the PIOP and the actual time of data sampling in the phase and amplitude head.

Special Reset Location:

The special reset location is mapped into address 40FFH . Reading or writing to this location causes an electrical reset of the digital circuitry of the phase and amplitude head.

ADC Data Block Structure:

The ADC data block contains the digitized values of the eight voltages measured each digitization cycle . It is structured as eight words, with each 12 bit value left justified in the 16 bit field.

The values are encoded in an offset binary format, with '0000'H representing -5 volts and 'FFFF'H representing +5 volts.

Table 3: ADC DATA BLOCK

WORD 0: Digitized system analog ground.
WORD 1: Digitized phase voltage sampled by phase electronics.
WORD 2: Digitized amplitude voltage sampled by the amplitude electronics, and -5 volt offset has been given.
WORD 3: Digitized temperature voltage measured in the ambient air of the phase and amplitude head. The system has a temperature coefficient of 10 mV/°K
WORD 4: Digitized DAC measurement prescaled by a factor of two. The 0-10 V DAC is scaled to 0-5 V by resistive divider.
WORD 5: Digitized +15 V supply voltage, as multiplied by .250 (i.e. +15 volts is scaled to 3.75 volts)
WORD 6: Digitized -15 V supply voltage, as multiplied by .250 (i.e. -15 volts is scaled to -3.75 volts)
WORD 7: Digitized differential voltage of +6 and -6 supplies. The voltage divider of ratio 0.537 : 0.463 is used. (i.e., voltages of +6 and -6 are scaled to 0.446 volts)

STATUS BLOCK

Status block contains recent values of many of the operational parameters. The Status Block is updated in memory, and the starting address of the block is moved to point the new block. Due to the nature of the asynchronous updating of the Status Block, there is a real and small 'collision' probability on CAM/C reads of the block. The collision rate is less than one percent, and is detected by invalid Q response from PIOP.

[STATUS_BLOCK Memory Map]

STATUS_BLOCK (0)	SOFT_REL_NUMBER
STATUS_BLOCK (1)	LINK_ID
STATUS_BLOCK (2)	TIME_STAMP
STATUS_BLOCK (3)	STATUS_W_DATA
STATUS_BLOCK (4)	TRIG_RATE/STA_ID_PDU
STATUS_BLOCK (5)	
STATUS_BLOCK (6)	CALCPHASE
STATUS-BLOCK (7)	AVE_CALCPHASF
STATUS-BLOCK (8)	VAMPLITUDE
STATUS-BLOCK (9)	AVE_VAMP
STATUS-BLOCK (A)	VMPED
STATUS-BLOCK (B)	AVE_VMPED
STATUS-BLOCK (C)	RAWVMPP
STATUS-BLOCK (D)	RAWVMMP
STATUS-BLOCK (E)	VMIX
STATUS-BLOCK (F)	LASTVMIX
STATUS-BLOCK (10)	MDAC
STATUS-BLOCK (11)	LASTMDAC
STATUS-BLOCK (12)	TEMP_NOW
STATUS-BLOCK (13)	TEMP_HOUR_AGO
STATUS-BLOCK (14)	PHASE_JITTER
STATUS-BLOCK (15)	AVE_PHASE_JITTER
STATUS-BLOCK (16)	AVE_AMP_JITTER
STATUS-BLOCK (17)	VAMP_OFF_OUT

STATUS-BLOCK (18)	
STATUS-BLOCK (19)	VP15
STATUS-BLOCK (1A)	VM15
STATUS-BLOCK (1B)	V6AU6
STATUS-BLOCK (1C)	VZERO
STATUS-BLOCK (1D)	VTEMP
STATUS-BLOCK (1E)	
STATUS-BLOCK (1F)	

STA-ID-PDU:

The station ID lights and the PDU channel are set by writing to location 5815H. The register is mapped as follows;

Bit	Write	
D7	16	} low true
D6	8	
D5	4	
D4	2	
D3	1	
D2-0	PDU channel	

TRIG-REG (Read/Write): Location 5816H controls trigger circuits.

Please refer to the schematic for complete Documentation. The register is mapped as follows;

Bit	Write
D0	Arm IDLE-TRIG
D1	Arm PDU-TRIG

D2
D3
D4 Enable IDLE_TRIG
D5 Disable IDLE_TRIG
D6 Enable MOD_TRIG
D7 Disable MOD_TRIG

Bit Read
D0 IDLE_TRIG came
D1 IDLE_TRIG enabled
D2 MOD_TRIG came
D3 MOD_TRIG enabled
D4 STANDBY PATTERN
D5 DMT0 (1 = OK-OK)
D6
D7

SOFTWARE DESCRIPTION

- The main routine of the program can be divided into three parts.

A) COLDSTART

COLDSTART is a preparation part of the program and is executed at the very beginning of the program. It initializes global variables, DMA, and interrupt conditions.

B) BACKGROUND

BACKGROUND is an infinite routine which forms a main body of the program. This routine gets command from control block and updates FTP, Status-block, and Status-word.

C) INTERRUPT ROUTINES

1. Get\$Raw\$Data , TYPE 18H, 19H

This routine does phase measurement and other voltage measurements.

The IDLE TRIGGER fires an interrupt type 18H. Under normal conditions type 18H interrupt will come at the machine trigger rate (360Hz). Type 19H interrupt is fired when both type 11H and type 18H interrupt condition occurs within the same clock pulse.

2. NMI_LINK_INTERRUPT , TYPE 2H

NMI interrupt is fired if no ACKnowledge is received from the remote head within 50 micro-seconds.

3. HEAD_INTERRUPT , TYPE 11H

Generated by hardware in the remote head.

• The main flow of the program is shown in Table 4.

Table 4: Main Flow of the Program MAIN ROUTINESA) COLDSTART:

⁴ Done by setting STA_ID_PDU=0H

⁵ call subroutine DMA_INIT

⁶ call subroutine HEAD_RESET

⁷ call INDICATOR(NO-LIGHT)

⁸ IB=TRIG_REG_LOC

* IB=RESET_NMI_LOC

¹⁰ all four functions done by [COMMAND\$REG=13H]

¹¹ all three functions done by [TRIG_REG= 91H]

a3

GOTO BACKGROUND

B) Background:

Light 2 ON [TRACE(2)]

Unload new control block of data from CAMAC

[CALL SERVICE_CAMAC]

Light 2,1 ON [TRACE(3)]

Load FTP data onto CAMAC if FTP is done

[call UPDATE_FTP]

a4

b4

C) Interrupt Routines:

1) Non_Maskable_Interrupt, TYPE 2H

Activated if the PIOP does not receive an acknowledge from the remote device for 50 microseconds.

NMI is cleared by reading from location 5817H

[RESET_NMI_LOC]

¹² used by CHECK_CABLE routine

¹³ BAD_CABLE gets logical value from CHECK_CABLE routine

¹ used in CHECK_CABLE ROUTINE

2) HEAD_INTERRUPT, TYPE 11H

Generated by hardware in the remote head.

It is enabled by writing a '1' and disabled by writing a '0' to the memory location 5817H

3)Get\$Raw\$Data

,TYPE 18H,19H

This interrupt routine takes care of phase,amplitude, and voltage measurements.

¹⁵ CLK_PERIODS is incremented at 360 Hz and is used to calculate time in clock routine

¹⁶ INTERRUPTED is set in Get\$Raw\$Data & Head_Interrupt routines. It is used to check if interrupt has occurred during the Status_Blk transfer in UPDATE_STATUS_BLK routine

* FLIP(ptr) is a short form of SIGNED((Signflip) XOR (UNSIGN(ADC(ptr)) where signflip = 8000H. This function converts binary value of ADC\$DATA\$BLOCK(ptr) into integer values in a linear fashion.

binary value Voltages of	Represented of FLIP (ptr)	binary value ADC\$DATA\$BLOCK	INTEGER values of FLIP(ptr)
FFF0	+5	7FF0	65520
FFE0	+4.997	7FE0	65504
.	.	.	.
.	.	.	.
.	.	.	.
8010	+0.003	3010	16
8000	+0.001	0000	0
7FF0	-0.001	FFF0	-16
.	.	.	.
.	.	.	.
.	.	.	.
0000	-5	0000	-65536

¹⁹ Given TRUE as a constant value changed by control_blk

²⁰ COUNTR is used to execute part of the routine every other pulse (at 180 Hz)

²¹ check for an overflow

²² LOCKED is a logical variable which indicates whether phase is locked or not.

²³ GET_AMP_PED is set to TRUE every one minute by CLOCK routine and reset by Get\$Raw\$data routine.

²⁴ digitization is initiated by setting bit 7 of COMMAND\$REG and resetting it

²⁵ To restore wobbler state

a) LOCKER routine

(2) TRACKER

²⁶ SLOPE = NOT (((LASTVMIX - VMIX)>0) XOR ((LASTMDAC - MDAC)>0))
TRUE if positive slope

D) OTHER SUBROUTINES1) CLOCK:

CLOCK routine takes care of Timing function.

* CLK_PERIODS is increased at 360 Hz and CLK_PERIODS_MIN has 60*180 as a constant value.

Hence one CLK_min is physically 30 seconds long

²⁹ get amplitude offset every minute

³⁰ TIME_STAMP is mapped as following

2) Update Status Blk:

Copies status block into CAMAC space if possible. Stores it to STATUS_BLK_1 and STATUS_BLK_2 alternately.

3) SERVICE CAMAC: Unloads new control of data block form CAMAC

SYSTEM PERFORMANCE

The linear collider under development at SLAC forces very severe constraints on the accelerator RF system.

Table 5: Jitter Detection Specialization

Sect_r	Phase Jitter	Amplitude Jitter
2	0.1 degrees	0.1 %
3	0.2 degrees	0.2 %
4-5	0.3 degrees	0.4 %
6-30	0.5 degrees	0.5 %

The measured performance of our single sector test system is shown in Table 6[†]

This result shows that our design satisfies the required precision. Please note that this accuracy depends on both software and hardware.

[†] measured at phase= 102.2513 , amplitude= 5.2503 V, Temp.=50.25 C
8/19/83

Table 6: Measured Performance of the Test System

Ave Phase Jitter	Ave Amplitude Jitter	Phase Range
0.0647°	0.0001 %	177.3409°-178.6234°

CONCLUSION

Modification of the program improved the system performance to some degree.

The major improvements are:

1. Wider measureable phase range.
(Typically, from $\pm 176.5^\circ$ to $\pm 178.2^\circ$)
2. Stabilizing the shake of digitized phase signal around $\pm 180^\circ$.
3. Reduced phase jitter around $\pm 180^\circ$.
4. Updating status block at every measurement.

The major modification of the program has been;

- a) Decrease in phase change in TRACKER routine ([DELTADAC]) by the factor of two.
- b) Introducing edge effect in TRACKER routine.
- c) Call of Update_Status_Block in the BACKGROUND routine.

The first improvement is due to a), second due to both a) and b), third due to b), and fourth due to c). However, we did not see significant improvement in ave_phase_jitter.³² Improvement in microwave circuitry is desired for even better performance

³² less than 10 %

ACKNOWLEDGEMENTS

Author wishes to thank Antonio Millich for his supervision. Keith Jobe, Mike Browne and Jim Harm for their help and support.

REFERENCES

1. 'Parallel Input/Output Processor Hardware Specifications'
Mike Browne , Keith Jobe SLAC No. 123-624
2. 'A New Control System For The SLAC Accelerator Klystrons For SLC'
Keith Jobe and Mike Browne SLAC-PUB-3077
3. 'Phase and Amplitude Detection System for the Stanford Linear Accelerator'
John Fox and H. Schwarz SLAC-PUB-3071
4. 'Microprocessor Controlled Phase Measurement System for 2856 MHz Pulses'
John Fox and H. Schwarz SLAC-PUB-2902
5. 'Linear Electronic Phase Shifter Design'
H. Schwarz PEP-NOTE 283

APPENDIX A

PL/M-86 COMPILER P1OPPROC

08/18/8

VAX/VMS PL/M-86 V2.1 COMPILATION OF MODULE P1OPPROC
 OBJECT MODULE PLACED IN SNYPHSE.OBJ
 COMPILER INVOKED BY: PLM21 SNYPHSE.TAB LARGE XREF

```

$LEFTMARGIN(7)
0010 $ OPTIMIZE(2)
0020 $ ROM
0030 $ DEBUG
0040 $ TYPE
0050 $ SYMBOLS
0060 $ NOINTVECTOR
0070
0080
1      0090 P1OPPROC:      DO:
0100 /*-----*/
0110 /*
0120 /*      REVISION DATA AND HISTORY
0130 /*
0140 /*-----*/
0150 /*      THIS IS A PROGRAM USED TO CONTROL PHASE AND AMPLITUDE
0160 /*
0170 /*      REVISED-BY-SUNG JOO YOO 8/18/1983
0180 /*      MOST OF THIS CODE HAS BEEN COPIED FROM
0190 /*      JOHN FOX AND KEITH JOBE ,AND THEY DESERVE CREDIT
0191 /*
0192 /* This version combines small and large version
0193 /* Takes care of edge effect in TRACKER routine and updates status_blk
0194 /*
0200 /*-----*/
2      0210 /*-----*/
1      0220 DECLARE SOFT_REL LITERALLY '1000B' ;
0230
0240 $ INCLUDE (FILENAME.INC)
/* Fully Qualified Vax Filename of Source Code */
3      1      0250 DECLARE VAX_FILE_NAME (*) BYTE DATA
/*
/*      ('DRA2:(STUDENT)SNYPHSE.PLM:5 18-AUG-1983 14:12:23.76');
0260 /*-----*/
0270 /*
0280 /*      GENERAL PURPOSE DECLARATIONS
0290 /*-----*/
4      1      0300
0310 DECLARE LOGICAL      LITERALLY      'BYTE',
0320 TRUE      LITERALLY      'OFFB',
0330 FALSE      LITERALLY      '000B',
0340 ENDDO      LITERALLY      'END;',
0350 ENDDIF      LITERALLY      ' ';
0360
5      1      0370 DECLARE MK_2_PROGRAM      LITERALLY      'FALSE',
0380 SB_PROGRAM      LITERALLY      'FALSE',
0390 PHASCOD      LITERALLY      'TRUE',
0400 STD_READ      LITERALLY      '(MK_2_PROGRAM OR SB_PROGRAM)',
0410 TRACE_MODE      LITERALLY      'TRUE',
0420 PRIMITIVE_MODE      LITERALLY      'TRUE',
0430

```



```

0440          TEST_MODE      LITERALLY      'TRUE';
6   1   0450      DECLARE PLUSFULL$SCALE  INTEGER DATA (7FFFH);
7   1   0460      DECLARE MINUSSFULL$SCALE INTEGER DATA (-32767);
0470
0480 * SET(TRUE)
0490 * RESET(FALSE)
0500 * RESET(MK_2_PROGRAM)
0510 * RESET(SB_PROGRAM)
0520 * SET(TEST_MODE)
0530 /* * RESET(SHORT) (implied, override with 'SET(SHORT)' qualifier) */
0540
0550
8   1   0560      DECLARE BAD_CABLE      LOGICAL,
0570          BAD_CABLE_TEST  LOGICAL,
0580          GOOD_TRIES      BYTE,
0590
0600          ERROR_SENT      LOGICAL,
0610          INTERRUPTED     LOGICAL,
0620
0630
0640          CLK_MINUTES     BYTE,
0650          CLK_PERIODS     WORD,
0660          OLD_CLK_PERIODS WORD,
0670          CLK_PERIODS_MIN  LITERALLY      '60*180',
0680          TIM_OUT_INTERVALS (*)  BYTE DATA
0690          (0,11,22,34,45,56), /* ONE SECOND STEPS */
0700
0710          (IN,JW,KW,LW)    WORD, /* FOR USE AS GENERAL PURPOSE */
0720          (IB,JB,EB,LD)    BYTE, /* REGISTERS */
0730          (II,JI,EI,LI)    INTEGER,
0740          (IL,JL,EL,LL)    LOGICAL,
0750          (IP,JP,KP,LP)    POINTER;
0760 /*
0770 /
0780 /
0790 /
0800 /*
0810      DECLARE (LOCKED,SLOPE,TRACKING_MODE,GET_AMP_PED) LOGICAL;
10  1   0820      DECLARE (TPASS,LPASS,PDAC0,
0830          WOBBLER, LASTWOBBLER, B_OFFSET,
0840          DELTADAC, STATUS)  BYTE;
11  1   0850      DECLARE WOBBLERMASK  BYTE DATA (40H);
12  1   0860      DECLARE FRESHUPDATESMASK  BYTE DATA (80H);
13  1   0870      DECLARE (CALCDAC,VAMP_OFFSET) INTEGER;
14  1   0900      DECLARE SIGNFLIP  LITERALLY '8000H';
15  1   0910      DECLARE HEAD(60H) INTEGER AT (4006H);
16  1   0920      DECLARE FEAD_RESULT_LDC BYTE AT (40FFH);
17  1   0930      DECLARE ADCOFFSET BYTE;
18  1   0940      DECLARE ADC_POINTER POINTER;
19  1   0950      DECLARE (ADCSDATASBLOCK BASED ADCSPINTER) (8) INTEGER;
20  1   0960      DECLARE (HEADTYPE,HEADSID,COMMANDSREG,STATUSREG
0970          ,DAC,CLOCKSCONTROLSDIVISOR,
0980          WOBBLESCONTROLSDIVISOR) BYTE AT (4000H);
0985 /*
0990      POINTERS FOR ADC BLOCK
21  1   0990      DECLARE ZEROSPTR  LITERALLY '0';
22  1   1000      DECLARE PHASESPTR  LITERALLY '1';
23  1   1010      DECLARE AMPSPTR  LITERALLY '2';

```

```

24 1 1020 DECLARE DACPTR LITERALLY '4';
25 1 1030 DECLARE TEMPPTR LITERALLY '3';
26 1 1040 DECLARE P158PTR LITERALLY '5';
27 1 1050 DECLARE M158PTR LITERALLY '6';
28 1 1060 DECLARE V6AU6SPTR LITERALLY '7';
29 1 1070 DECLARE RED_LIGHT LITERALLY '1';
30 1 1080 DECLARE GREEN_LIGHT LITERALLY '2';
31 1 1090 DECLARE NO_LIGHT LITERALLY '0';
1100 /*=====
1110 / PLOP SYSTEM LOCATIONS
1120 /
1130 /*=====
32 1 1140 DECLARE STATUS_BLK_LEN LITERALLY '32';
1150 CTL_BLK_LEN LITERALLY '16';
1160 FTP_BLK_LEN LITERALLY '(FTP_SAMPLES+3)';
33 1 1170 DECLARE STATUS_BLK (STATUS_BLK_LEN) WORD;
1180
34 1 1190 DECLARE SOFT_REL_NUMBER WORD AT (@STATUS_BLK(0)),
1200 LINK_ID WORD AT (@STATUS_BLK(1)),
1210 TIME_STAMP WORD AT (@STATUS_BLK(2)),
1220 STATUS_W_DATA WORD AT (@STATUS_BLK(3)),
1230 ( TRIG_RATE, STA_ID_PDU )
1240 BYTE AT (@STATUS_BLK(4));
1250
35 1 1260 DECLARE DMA_LOC (10H) BYTE AT (3800H),
1270
1280 HEAD_LOC (100H) BYTE AT (4000H),
1290
1300 LINK_ID_LOC WORD AT (CHEAD_LOC(00)),
1310
1320 ERROR_WORD_LOC WORD AT (3810H),
1330 STATUS_W_LOC WORD AT (3812H),
1340
1350 STATUS_FLG_LOC BYTE AT (3814H),
1360 STA_ID_PDU_LOC BYTE AT (3815H),
1370 TRIG_REC_LOC BYTE AT (3816H),
1380 RESET_RMI_LOC BYTE AT (3817H);
1390
1400
1410 /* The following addresses are also encoded in DMA_INIT and
1420 UPDATE_STATUS_BLK */
1430
1440
1450
36 1 1460 DECLARE STATUS_BLK_1_LOC (20H) WORD AT (3000H),
1470 STATUS_BLK_2_LOC (20H) WORD AT (3040H),
1480
1490 CTL_BLK_LOC (10H) WORD AT (3000H),
1500
1510 FTP_BLK_LOC (43H) WORD AT (30A0H),
1520 FTP_ARRAY (40H) INTEGER AT (6FTP_BLK_LOC(3)),
1530
1540 FTP_DATA INTEGER, /* TEMP. REGISTER */
1550 FTP_SAMPLE_R0 BYTE,
1560 FTP_ON LOGICAL,
1570 ( FTP_CHAN, FTP_DELAY, FTP_STEP )
1580 WORD AT (6FTP_BLK_LOC(0)),

```

```

1590      FTP_MODE      BYTE,
1600      FTP_CURRENT_DELAY  WORD,
1610      FTP_SAMPLES   LITERALLY '64',
1620      FTP_PED_DELAY LITERALLY '0',/* 2 uSEC */
1630      FTP_PED_VALUE  INTEGER,
1640      FTP_PEDISTAL (20H) INTEGER,
1650      FTP_PED_CNT    BYTE;
37 1      DECLARE HEAD_ECHO_LOC      BYTE      AT (@HEAD_LOC(01)),
1670
1680      HEAD_DAC_LOC      WORD      AT (@HEAD_LOC(12)),
1690      HEAD_DRIVE_LOC    WORD      AT (@HEAD_LOC(40H)),
1700      HEAD_FTP_CHAN_LOC BYTE      AT (@HEAD_LOC(12)),
1710      HEAD_FTP_DELAY_LOC BYTE      AT (@HEAD_LOC(12)),
1720      HEAD_STEP_MOTOR   BYTE      AT (@HEAD_LOC(4FH)),
1730      HEAD_WATER_LOC    BYTE      AT (@HEAD_LOC(48H)),
1740
1750      HEAD_ADC_CHAN_LOC  BYTE      AT (@HEAD_LOC(12)),
1760      HEAD_ADC_LOC      INTEGER AT (@HEAD_LOC(12));
1770
1780
1790 /*=====
1800 /
1810 /      MAPPING OF STATUS BUFFER FOR CAMAC FOR MK2 PROGRAM
1820 /
1830 /*=====
38 1      DECLARE
1850      FE_POWER      INTEGER      AT (@STATUS_BLK(06H)),
1860      FE_POWER_TH   INTEGER      AT (@STATUS_BLK(07H)),
1870      RE_POWER      INTEGER      AT (@STATUS_BLK(08H)),
1880      RE_POWER_TH   INTEGER      AT (@STATUS_BLK(09H)),
1890      BEAM_VOLTS     INTEGER      AT (@STATUS_BLK(0AH)),
1900      BEAM_VOLTS_TH INTEGER      AT (@STATUS_BLK(0BH)),
1910      BEAM_CUR       INTEGER      AT (@STATUS_BLK(0CH)),
1920      BEAM_CUR_TH   INTEGER      AT (@STATUS_BLK(0DH)),
1930      DRIVE_REQ      WORD        AT (@STATUS_BLK(0EH)),
1940      DRIVE          WORD        AT (@STATUS_BLK(0FH)),
1950      PHASE          INTEGER      AT (@STATUS_BLK(10H)),
1960      PHASE_REQ      INTEGER      AT (@STATUS_BLK(11H)),
1970      MOD_STATUS     WORD        AT (@STATUS_BLK(12H)),
1980      ( H2O_FAULT_CNT, H2O_FAULT_CNT_OLD )
1990      ( FAULT_COUNT, FAULT_COUNT_OLD )
2000      ( STATUS_BLK(13H)),
2010      ( STATUS_BLK(14H)),
2020      ( STATUS_BLK(15H)),
2030      ( STATUS_BLK(16H)),
2040      ( STATUS_BLK(17H)),
2050      ( STATUS_BLK(18H)),
2060      ( STATUS_BLK(19H)),
2070      ( STATUS_BLK(1AH)),
2080      ( STATUS_BLK(1BH)),
2090      ( STATUS_BLK(1CH)),
2100 /*      DECLARATIONS FOR PHASE AND AMPLITUDE HEAD
2110      STATUS BUFFER
2120      DECLARE
39 1      CALCPHASE      INTEGER      AT (@STATUS_BLK(06H)),
2130      AVE_CALCPHASE    INTEGER      AT (@STATUS_BLK(07H)),
2140      VANPLITUDE       INTEGER      AT (@STATUS_BLK(0CH)),
2150

```

```

2160 AVE_VAMP INTEGER AT (@STATUS_BLK(09H)),
2170 VMPED INTEGER AT (@STATUS_BLK(0AH)),
2180 AVE_VMPED INTEGER AT (@STATUS_BLK(0BH)),
2190 RAMVMPF INTEGER AT (@STATUS_BLK(0CH)),
2200 RAMVMPF INTEGER AT (@STATUS_BLK(0DH)),
2210 VMIX INTEGER AT (@STATUS_BLK(0EH)),
2220 LASTVMIX INTEGER AT (@STATUS_BLK(0FH)),
2230 MDAC INTEGER AT (@STATUS_BLK(10H)),
2240 LASTMDAC INTEGER AT (@STATUS_BLK(11H)),
2250 TEMP_FOW INTEGER AT (@STATUS_BLK(12H)),
2260
2270 TEMP_HOUR_AGO INTEGER AT (@STATUS_BLK(13H));
40 1 2280 DECLARE
2290 PHASE_JITTER INTEGER AT (@STATUS_BLK(14H)),
2300 AVE_PHASE_JITTER INTEGER AT (@STATUS_BLK(15H)),
2310 AMP_JITTER INTEGER AT (@STATUS_BLK(16H)),
2320 AVE_AMP_JITTER INTEGER AT (@STATUS_BLK(17H)),
2330 VAMP_OFF_OUT INTEGER AT (@STATUS_BLK(18H)),
2332 VF15 INTEGER AT (@STATUS_BLK(1AH)),
2334 VM15 INTEGER AT (@STATUS_BLK(1BH)),
2336 V6AU6 INTEGER AT (@STATUS_BLK(1CH)),
2338 VZ10 INTEGER AT (@STATUS_BLK(1DH)),
2340 VTEMP INTEGER AT (@STATUS_BLK(1EH));
2350 /*=====
2360 /
2370 / ERROR WORDS FOR PIOP
2380 /
2390 /*=====
41 1 2400
2410 DECLARE SYS_ERR_DMT0 LITERALLY '0000H',
2420 SYS_ERR_Y_RAM LITERALLY '0001H',
2430 SYS_ERR_S_RAM LITERALLY '0002H',
2440 SYS_ERR_CTL_CSUM LITERALLY '0003H',
2450 SYS_ERR_FTP_CSUM LITERALLY '0004H',
2460 SYS_ERR_LINK LITERALLY '0005H',
2470 SYS_ERR_LINK_ID LITERALLY '0006H',
2480 SYS_ERR_LINK_ID_TYPE LITERALLY '0007H',
2490 SYS_ERR_CTL_BAD LITERALLY '0008H';
2500 /*=====
2510 /
2520 / ERROR WORDS FOR MK-2
2530 /
2540 /
2550 /*=====
42 1 2560
2570 DECLARE MK2_ERR_VACUUM LITERALLY '1001H',
2580 MK2_ERR_WATER LITERALLY '1002H',
2590 MK2_ERR_BEAM_VOLTS LITERALLY '1003H',
2600 MK2_ERR_BEAM_CUR LITERALLY '1004H',
2610 MK2_ERR_RE_POWER LITERALLY '1005H',
2620 MK2_ERR_LOCAL_ON LITERALLY '1006H',
2630 MK2_ERR_REMOTE_ON LITERALLY '1007H',
2640 MK2_ERR_LOCAL_OFF LITERALLY '1008H',
2650 MK2_ERR_REMOTE_OFF LITERALLY '1009H',
2660 MK2_ERR_DRIVE LITERALLY '100AH';
2670
2680

```

```
2690
2700 /*-----*
2710 /
2720 /      ERROR WORDS FOR SUB_BOOSTER
2730 /
2740 /*===== *
2750 /
2760 /
2770 /*-----*
2780 /
2790 /      ERROR WORDS FOR STD_HEAD
2800 /
2810 /*===== *
2820 /
2830 /
2840 /*===== *
2850 /*===== *
2860 /*===== *
2870 * EJECT
```

```
2880 /*=====*/
2890
43 1 2900 TRACE:      PROCEDURE (DELTA);
2910
2920 /* Writes the data in argument to the front panel */
2930
44 2 2950      DECLARE DELTA      BYTE;
2960
45 2 2970      IF TRACE_MODE THEN
46 2 2980          STA_ID_PDU_LOC = (STA_ID_PDU AND 07H) OR
2990          SHL((NOT DELTA),3);
3000      ENDIF
3010
47 2 3020      RETURN;
48 2 3030 END TRACE;
3040 $EJECT
```

```

3050 /*=====*/
49 1 3060 ERROR_SEND: PROCEDURE (ERROR$WORD) REENTRANT;
3080
3090 /* This routine sends an error word to the error-word buffer for
3100 / retrieval by CAMAC. If LAM is enabled, L is asserted. */
50 2 3110
3120 DECLARE ERROR$WORD WORD;
3130
3140 /* ONLY THE FIRST ERROR STAYS IN THE CAMAC BUFFER (FOLLOWING CAMAC READ ) */
51 2 3150 IF (STATUS_FLG_LOC AND 08H) = 00H THEN
52 2 3160 DO;
53 3 3170 ERROR_WORD_LOC = ERROR$WORD;
54 3 3180 STATUS_FLG_LOC = 07H; /* ARM ERROR FLAG */
55 3 3190 ENDDO
3200 ENDIF
3210
56 2 3220 ERROR_SENT = TRUE; /* AT LEAST WE TRIED !!! */
3230
57 2 3240 RETURN;
58 2 3250 END ERROR_SEND;
3260
3270 $ EJECT

```

```

3280 /*=====*/
3290 /*=====*/
59 1 3300 CHECK_SUM:      PROCEDURE (BLOCK$PTR, W$COUNT) LOGICAL;
3310
3320
3330 /* C_SUM Returns logic representing the status of the block.
3340 /* W$COUNT is a byte descriptor of the length of the block (in words!)
3350 /* not including the CSUM vector.  ie:
3360
3370 /   IF CHECK_SUM(@CTL_BLK_LOC,3) THEN ...   */
60 2 3380
3390     DECLARE BLOCK$PTR      POINTER,
61 2 3400     W$COUNT              BYTE;
3410     DECLARE BLOCK BASED BLOCK$PTR (1) WORD;
3420
62 2 3430     IW = 0;
63 2 3440 LOOP:   DO IB = 0 TO W$COUNT;   /* W$COUNT EXCLUDES CSUM VECTOR */
64 3 3450     IW = IW + BLOCK(IB);
65 3 3460     END LOOP;
3470
66 2 3480     IW = IW - BLOCK(W$COUNT + 1);
67 2 3490     IF IW = 0 THEN RETURN TRUE;     /* WE PASSED */
69 2 3500     RETURN FALSE;                  /* WE FAILED */
70 2 3510 END CHECK_SUM;
3520
3530 $ EJECT

```



```
3540 /*=====*/
3550
71 1 3560 CTL_CSUM:      PROCEDURE (I) LOGICAL;
3570
3580 /* To allow more code efficient coding of check_sum for the
3590 /  ctl_block uses. */
3600
72 2 3610      DECLARE I      BYTE;
3620
73 2 3630      IF CHECK_SUM(@CTL_BLK_LOC,I) THEN
74 2 3640          RETURN FALSE; /*BINGO ITS OK-OK */
3650      ENDIF
3660
75 2 3670      CALL ERROR_SEND(SYS_ERR_CTL_CSUM);
76 2 3680      RETURN TRUE;
77 2 3690 END CTL_CSUM;
3700
3710 • EJECT
```

```

3720 /*=====*/
3730 /*=====*/
78 1 3740 DMA_INIT:      PROCEDURE;
3750
3760 /* Initializes the DMA channel device in the PIOP. An INTEL manual
3770 is required to understand this section of the code.
3780
3790 Look under INTEL 8237/8237A HIGH PERFORMANCE PROGRAMMABLE
3800 DMA CONTROLLER.
3810
3820 */
3830
79 2 3840 DMA_LOC(0DH) = TRUE;      /* MASTER CLEAR      */
80 2 3850 DMA_LOC(08H) = 00000000B; /* COMMAND REG      */
81 2 3860 DMA_LOC(0BH) = 00010000B; /* MODE REGISTER 0  */
82 2 3870 DMA_LOC(0BH) = 00010010B; /*                   */
83 2 3880 DMA_LOC(0BH) = 00010100B; /*                   */
84 2 3890 DMA_LOC(0BH) = 00010110B; /*                   */
3900
85 2 3910 DMA_LOC(0CH) = TRUE;      /* CLEAR F/L FF      */
3920
3930
3940
86 2 3950 DMA_LOC(00H) = 80H;      /* CONTROL BLOCK      */
87 2 3960 DMA_LOC(00H) = 50H;
3970
88 2 3980 DMA_LOC(01H) = LOW(2*CTL_BLK_LEN-1);
89 2 3990 DMA_LOC(01H) = HIGH(2*CTL_BLK_LEN-1);
4000
90 2 4010 DMA_LOC(0AH) = 00000000B;
4020
4030
91 2 4040 DMA_LOC(02H) = 0A0H;      /* FTP BLOCK          */
92 2 4050 DMA_LOC(02H) = 50H;
4060
93 2 4070 DMA_LOC(03H) = LOW(2*FTP_BLK_LEN-1);
94 2 4080 DMA_LOC(03H) = HIGH(2*FTP_BLK_LEN-1);
4090
95 2 4100 DMA_LOC(0AH) = 00000001B;
4110
4120
4130 /*-----*/ /* STATUS BLOCK */
4140
96 2 4150 DMA_LOC(05H) = LOW(2*STATUS_BLK_LEN-1);
97 2 4160 DMA_LOC(05H) = HIGH(2*STATUS_BLK_LEN-1);
4170
98 2 4180 DMA_LOC(0AH) = 00000110B; /* TURN OFF CRANNEL */
4190
4200
99 2 4210 DMA_LOC(06H) = 01; /* INTEL CAUTIONS US TO INIT UNUSED CHANNELS */
100 2 4220 DMA_LOC(06H) = 00;
4230
101 2 4240 DMA_LOC(07H) = 01;
102 2 4250 DMA_LOC(07H) = 00;
4260
103 2 4270 DMA_LOC(0AH) = 00000111B; /* TURN IT OFF FOR GOOD */
4280

```

AL/N-86 COMPILER PLOPPROG

08/18

```
104 2      4290
105 2      4300 RETURN;
      4310 END DMA_INIT;
      4320
      4330 * EJECT
```

```
4340 /*=====*/
4350
106 1 4360 NMI_LINK_INT: PROCEDURE INTERRUPT 2;
4370
4380 /* This routine is called by the NMI (Non-Maskable Interrupt) if the
4390 / plop did not receive an acknowledge from the remote device following
4400 / an access within some interval. (typically 50 microseconds). */
4410
107 2 4420 BAD_CABLE_TEST = TRUE; /* USED BY CHECK_CABLE */
4430
108 2 4440 IB = RESET_NMI_LOC; /* DUMMY VALUE. UNHANGS NMI INTERRUPT. */
4450
109 2 4460 IF BAD_CABLE THEN RETURN; /* WE ALREADY KNOW ABOUT IT. */
4470
111 2 4480 CALL ERROR_SEND(SYS_ERR_LINK); /* ELSE TELL CAMAC */
112 2 4490 BAD_CABLE = TRUE; /* AND TELL SYSTEM */
4500
113 2 4510 RETURN;
114 2 4520 END NMI_LINK_INT;
4530
4540 ■ EJECT
```

```

4550 /*=====*/
4570 /*=====*/
115 1 4580
4590 CLOCK: PROCEDURE;
4600
4610 /* This routine takes care of the time related functions. These
4620 / include proper track of minutes and hours to reset fault
4630 / counters, etc. Note that this is a background task, and may
4640 / miss several cycles prior to re-appearing (at times). */
4650
116 2 4660 DECLARE TEMP$PERIODS WORD;
4670
117 2 4680 IF CLK_PERIODS >= CLK_PERIODS_MIN THEN
118 2 4690 DO:
119 3 4700 DISABLE; /* SO WE DO NOT GET INTERRUPTED */
120 3 4710 TEMP$PERIODS=(CLK_PERIODS := CLK_PERIODS - CLK_PERIODS_MIN);
121 3 4720 ENABLE;
122 3 4730 CLK_MINUTES = CLK_MINUTES + 1;
123 3 4740 GET_AMP_PED=TRUE; /* ONCE A MIN GET THE AMP PEDESTAL */
124 3 4750 IF CLK_MINUTES >= 60 THEN /* BRING IN NEW HOUR */
125 3 4760 DO:
126 4 4770 CLK_MINUTES = CLK_MINUTES - 60;
4780 /* RESET FAULT COUNT HERE, ETC */
4790 /*
4800 WE NOTE THE TEMPERATURE AT THIS POINT AND SAVE IT */
4810
127 4 4820 TEMP_HOUR_AGO=TEMP_NOW;
4830 /* GREAT TO ALLOW TIMEOUT TO WORK */
4840 /* NOTE - IF TIMOUTED ANYWAY, GIVES 1 SECOND REGARDLESS
4850 / OF REMAINING TIME (TYP. 0 - 3 SECONDS.) */
128 4 4860 ENDDO
4870 ENDDIF
29 3 4880 ENDDO
4890 ENDDIF
4900
130 2 4910 TIME_STAMP = SHL(DOUBLE(CLK_MINUTES),10) OR
4920 TEMP$PERIODS; /* MIN - CLOCKS */
4930 /* NOTE !!! 5 ABOVE = 4 FOR 180 PPS OPERATION !!! */
131 2 4940
132 2 4950 RETURN;
133 2 4960 END CLOCK;
4970
4980 $ EJECT

```

PL/M-86 COMPILER PIOPPROC

```

4990 /******
5000
134 1 5010 BLINK: PROCEDURE;
5020
5030 /* BLINK front panel lights. */
5040
135 2 5050           DISABLE;
5060
136 2 5070           IB = 0;
137 2 5080 L1:       CALL TIME(2500);       /* 1/4 SECOND STEPS */
138 2 5090       STA_ID_PDU_LOC = ( IB := IB - 8H );
139 2 5100       GOTO L1;
140 2 5110 END BLINK;
5120
5130 * EJECT

```

```

141 1 5140 /*=====*/
5150
5160 UPDATE_STATUS_WORD: PROCEDURE;
5170
5180 /* Provides a close to current copy of the status word.
5190 / Hits the DMT0 one shot. */
5200
142 2 5210 DECLARE OLDSCLK_PERIODS WORD;
5220
143 2 5230 IF (NOT TEST_MODE) AND
5240 (OLDSCLK_PERIODS = CLK_PERIODS) AND
5250 ((STATUS_FLG_LOC AND 04H) = 04H) THEN
144 2 5260 RETURN;
5270 ENDF
5280
145 2 5290 IW = 0;
5300
146 2 5310 IF MK_2_PROGRAM THEN
147 2 5320 DO;
5330 /*
5340 I HAVE COMMENTED OUT THE K. JOBE STATUS WORD
5350 AND LEFT IT AS A TEMPLATE FOR THE PHASE HEAD
5360 STATUS WORD.
5370 IF DRIVE = DRIVE_REQ THEN IW = 1W OR 1B;
5380 IF KLYSTRON_ON THEN IW = 1W OR 10B;
5390 IF KLYSTRON_GO THEN IW = 1W OR 100E;
5400 IF FE_POWER >= FE_POWER_TH THEN IW = 1W OR 1000E;
5410 IF SLED_TUNED THEN IW = 1W OR 10000B;
5420 IF BAD_CABLE THEN IW = 1W OR 100000E;
5430 IF TIM_OUT THEN IW = 1W OR 1000000B;
5440 IF WATER_FAULT THEN IW = 1W OR 10000000E;
5450 IF VACUUM_FAULT THEN IW = 1W OR 100000000B;
5460 IF DRIVE_FAULT THEN IW = 1W OR 1000000000E;
5470 AND BACK TO REAL CODE HERE */
148 3 5480 ENDDO
5490 ENDF
5500
149 2 5510 IF TEST_MODE THEN
150 2 5520 IW = 0AA55H;
5530 ENDF
5540
151 2 5550 OLDSCLK_PERIODS = CLK_PERIODS;
152 2 5560 STATUS_W_DATA = IW;
153 2 5570 DISABLE;
154 2 5580 STATUS_FLG_LOC = 02H;
155 2 5590 STATUS_W_LOC = 1W;
156 2 5600 STATUS_FLG_LOC = 06H;
157 2 5610 ENABLE;
5620
158 2 5630 RETURN;
5640
159 2 5650 END UPDATE_STATUS_WORD;
5660
5670 * EJECT

```

```

5680 /*=====*/
5690
160 1 5700 UPDATE_STATUS_BLK:      PROCEDURE;
5710
5720 /* Copies status block into CAMAC space, if possible.
5730 /
5740 / Note that there is a small collision probability on a CAMAC
5750 / read durring transfer.
5760 /
5770 / The flag BLOCK2 is used to indicate which block is available
5780 / to CAMAC at this time. BLOCK2 = True indicates that Status_
5790 / Block_1_Loc is available for updating.
5800 /
5810 / Note that on power up -- BLOCK2 is either T or F. It doesn't
5820 / matter.
5830 */
5840
161 2 5850 DECLARE BLOCK2 LOGICAL,
5860 BADSTRIESLIMIT LITERALLY '4000',
5870 BADSTRIES WORD;
5880
162 2 5890 INTERRUPTED = FALSE; /* WE WANT A CLEAN COPY */
5900
163 2 5910 /* Copy STATUS_BLOCK into unused CAMAC Block */
164 2 5920 IF BLOCK2
165 2 5930 THEN CALL MOVW(@STATUS_BLK,@STATUS_BLK_1_LOC,STATUS_BLK_LEN);
5940 ELSE CALL MOVW(@STATUS_BLK,@STATUS_BLK_2_LOC,STATUS_BLK_LEN);
5950 ENDIF
5960
60 2 5970 /* If we were interrupted, try again next time */
5980 IF INTERRUPTED THEN RETURN;
5990
6000 /* SEE IF CAMAC IS IN PROGRESS (AS BEST WE CAN) */
6010
166 2 6020 IB = LOW(2*STATUS_BLK_LEN-1);
169 2 6030 JB = HIGH(2*STATUS_BLK_LEN-1);
6040
170 2 6050 IF BLOCK2
171 2 6060 THEN IW = 500CH; /* S_B_1_LOC LOCATION */
172 2 6070 ELSE IW = 504CH; /* S_B_2_LOC LOCATION */
6080 ENDF
6090
173 2 6100 DMA_LOC(0CH) = TRUE; /* CLEAR F/L FF */
6110
174 2 6120 BADSTRIES = BADSTRIES + 1;
6130
175 2 6140 IF BADSTRIES > BADSTRIESLIMIT THEN
176 2 6150 IF DMA_LOC(05H) = IB THEN
177 2 6160 IF DMA_LOC(05H) = JB THEN
178 2 6161 ; /* -- THIS IS THE OKOK CASE -- */
179 2 6162 ELSE
6170 RETURN;
6180 ENDF
6182 ELSE
6183 RETURN;
6190 ENDF
6200 ENDF

```



```

181  2      6210
182  2      6220      DMA_LOC(0AH) = 00000110B; /* TURN OFF CHANNEL */
183  2      6230
184  3      6240 /* TO DESTROY A EMBARRASING FAILURE MODE */
185  3      6250
186  3      6260 /* It is possible to check for CAMAC activity, and reset the
187  3      6270 /* base address while CAMAC just extracts the first entry. This
188  2      6280 /* would result a valid Q, excess byte count, and a shifted read
189  2      6290 /* block. This will abort the reset or force invalid Q
190  2      6300 /* (while increasing the collision probability) */
191  2      6310
192  2      6320      IF (DMA_LOC(05H) <> 1B) AND (BADSTRIES < BADSTRIESLIMIT) THEN
193  2      6330      DO;
194  2      6340          DMA_LOC(0CH) = TRUE;
195  2      6350          DMA_LOC(0AH) = 00000010B;
196  2      6360          RETURN;
197  2      6370      ENDDO
198  2      6380      ENDIF
199  2      6390
200  2      6400 /* Flip Blocks 1 <--> 2. */
201  2      6410      DMA_LOC(0CH) = TRUE;
202  2      6420      DMA_LOC(04H) = LOW(1W);
203  2      6430      DMA_LOC(04H) = HIGH(1W);
204  2      6440      DMA_LOC(0AH) = 00000010B; /* TURN ON CHANNEL */
205  2      6450      BADSTRIES = 0;
206  2      6460
207  2      6470      BLOCK2 = NOT BLOCK2;
208  2      6480
209  2      6490      RETURN;
210  2      6500 ENL UPDATE_STATUS_BLK;
211  2      6510
212  2      6520 $ EJECT

```

```

        6530 /*=====*/
        6540
196 1    6550 UPDATE_FTP:    PROCEDURE;
        6560
        6570 /* LOADS FTP DATA INTO CAMAC, IF FTP IS DONE. */
        6580
197 2    6590            IF FTP_SAMPLE_NO < FTP_SAMPLES THEN RETURN;
        6600
        6610
        6620 /* WE WANT TO FALL BACK INTO THE PEDISTAL COLLECTION MODE */
        6630 /* MODE 0 = TRANSITION MODE
        6640 /      1 = TIME DOMAIN FTP
        6650 /      2 = DRIVE DOMAIN FTP
        6660 /      3 = UNUSED
        6670 */
199 2    6680            FTP_MODE = 0;
200 2    6690            FTP_CURRENT_DELAY = FTP_PED_DELAY;
        6700 /*      HEAD_FTP_CHAN_LOC = HEAD_FTP_GROUND;    */
201 2    6710            FTP_ON = FALSE;
202 2    6720            FTP_SAMPLE_NO = 0;
        6730
        6740 /* TURN ON FTP BUFFER */
        6750
203 2    6760            STATUS_FLG_LOC = 05E;                /* ENABLE CAMAC READS OF FTP */
        6770
204 2    6780            RETURN;
205 2    6790 END UPDATE_FTP;
        6800
        6810 $ EJECT
```

```

6820 /*=====*/
6830
206 1 6840 TEST_LINK:      PROCEDURE LOGICAL;
6850
6860 /* Tests LINK_ID.
6870 / RETURNS TRUE if OK-OK
6880 / FALSE if not.
6890 / Assumes a new head only if status block contains 0000H for head_id.
6900 /
6910 / High order bit MUST match program mode. */
207 2 6920      IW = LINK_ID_LOC;
208 2 6930      IF BAD_CABLE THEN RETURN FALSE;
6940
210 2 6950      IF LINK_ID = 0 THEN LINK_ID = IW;
6960 /*      I HAVE COMMENTED OUT THE ERROR CALL HERE AS
6970      CALL ERROR_SEND(SYS_ERR_LINK_ID_TYPE);
6980      RETURN FALSE;
6990      I DON'T HAVE THE ID TABLE JUST YET      */
212 2 7000 L1:      IF IW = LINK_ID THEN RETURN TRUE;
7010
214 2 7020      CALL ERROR_SEND(SYS_ERR_LINK_ID);
215 2 7030      RETURN FALSE;
7040
216 2 7050 END TEST_LINK;
7060
7070 $ EJECT
```

```

217 1 7080 /*=====*/
      7090
      7100 CHECK_CABLE: PROCEDURE:
      7110
      7120 /* This checks the cable to the external device by sending patterns to
      7130 / and echo port at the head and checking for accuracy. If the cable
      7140 / was bad, this test must be passed several times prior to setting
      7150 / BAD_CABLE FALSE. */
      7160
218 2 7170 DECLARE PATTERN BYTE;
      7180
219 2 7190 BAD_CABLE_TEST = FALSE; /* OUR PRIVATE FLAG TO SEE IF THE
      7200 / CABLE BOMED AGAIN. */
220 2 7210 IF BAD_CABLE AND (GOOD_TRIES = 0) THEN
221 2 7220 GOOD_TRIES = 500; /* IF CABLE WAS BAD, INSIST ON
      7230 / 500 TESTS PASSED. */
222 2 7240 LOOP: DO IB = PATTERN TO PATTERN + 10H; /* 16 BYTE SUB-PATTERN */
      7250
223 3 7260 IF STD_HEAD THEN
224 3 7270 DO;
225 4 7280 HEAD_ECHO_LOC = IB; /* SEND PATTERN */
226 4 7290 IF HEAD_ECHO_LOC = IB THEN /* RECEIVED OK */
227 4 7300 IF NOT BAD_CABLE_TEST THEN /*AND WE DIDN'T TIMOU */
228 4 7310 GOTO L1;
      7320 ENDDIF
      7330 ENDDIF
      7340 ENDDO
229 4 7350 ENDDIF
      7360 IF PHASCOD THEN
230 3 7370 DO;
231 3 7380 HEAD(4) = INT(IB); /* SEND PATTERN */
232 4 7390 IF HEAD(4) = INT(IB) THEN /* RECEIVED OK */
233 4 7400 IF NOT BAD_CABLE_TEST THEN /*AND WE DIDN'T TIMOU */
234 4 7410 GOTO L1;
235 4 7420 ENDDIF
      7430 ENDDIF
      7440 ENDDO
      7450 ENDDIF
236 4 7460 /* IF WE ARE HERE WE FAILED ! */
237 3 7470 IF NOT BAD_CABLE THEN
238 3 7480 CALL ERROR_SEND(SYS_ERR_LINK);
      7490 ENDDIF
239 3 7500 BAD_CABLE = TRUE;
240 3 7510 RETURN; /* FAILED TEST */
241 3 7520 L1: END LOOP;
      7530
242 2 7540 PATTERN = IB;
      7550
243 2 7560 IF GOOD_TRIES > 0 THEN /* THIS IS THE CASE WHERE WE ARE */
244 2 7570 DO; /* PROVING A PREVIOUSLY BAD LINK */
245 3 7580 IF (GOOD_TRIES := GOOD_TRIES - 1) = 0 THEN
246 3 7590 DO;
247 4 7600 BAD_CABLE = FALSE;
248 4 7610 IL = TEST_LINK; /* A GOOD TIME TO TEST THE HEAD ID */
249 4 7620 ENDDO
      7630 ENDDIF
250 3 7640 ENDDO

```

```
251 2      7650      ENDIF
252 2      7660      RETURN;          /* PASSED TEST */
253 1      7670 END CHECK_CABLE;
254 2      7680 HEAD_RESET;  PROCEDURE;
255 2      7690          18=HEAD_RESET_LOC;
256 2      7700      RETURN;
          7710 END HEAD_RESET;
          7720 * EJECT
```

PL/N-86 COMPILER PIOFPROC

```

257 1 7730 /*****
258 2 7740 INDICATOR:  PROCEDURE (MODE);
259 2 7750          DECLARE (MODE,IB,ITEMP) BYTE;
260 2 7760          IB= (COMMAND$REG AND 9FH);
261 2 7770          ITEMP= SHL( (MODE AND 03H),5);
262 2 7780          COMMAND$REG= IB OR ITEMP;
263 2 7790          RETURN;
      7800 END INDICATOR;
      7810 $ EJECT

```

```
7820 /*=====*/
7830
264 1 7840 PHASECALC:      PROCEDURE;
265 2 7850     IF (LASTVMIX-VMIX)=0 THEN CALCDAC=LASTMDAC; ELSE
267 2 7860     CALCDAC= LASTMDAC +
7870         SAL((SAR(LASTVMIX,4)*SAR(MDAC-LASTMDAC,4))/
7880         (SAR(LASTVMIX-VMIX,4)),4); /* calculate Vzero */
268 2 7890     IF (SLOPE) THEN CALCPHASE=CALCDAC; /* add 180 if slope < 0 */
270 2 7900     ELSE CALCPHASE= SIGNED(UNSIGN(CALCDAC) + 8000H) ;
271 2 7910     PHASE_JITTER=CALCPHASE - AVE_CALCPHASE;
272 2 7920     AVE_CALCPHASE= CALCPHASE/2 + AVE_CALCPHASE/2;
273 2 7930     AVE_PHASE_JITTER= IABS(PHASE_JITTER)/2 + AVE_PHASE_JITTER/2;
274 2 7940     RETURN;
275 2 7950     END PHASECALC;
7960
7970 * EJECT
```

```

7980 /*=====*/
7990
276 1 8000 TRACKER: PROCEDURE;
277 2 8001 IF (TPASS<>1) THEN /* ovf occurred or first entry? */
278 2 8002 DO:
279 3 8003 LASTVMIX=VMIX;
280 3 8004 LASTMDAC=MDAC;
281 3 8005 IF (TPASS = 0) THEN
282 3 8006 DAC=DAC+DELTADAC; /* inc ovf , increment further */
283 3 8007 ELSE
284 3 8008 DAC= DAC - DELTADAC; /* dec ovf , decrement further */
285 3 8009 TPASS=1;
286 3 8010 RETURN;
287 2 8011 END;
288 2 8012 SLOPE=NOT(((LASTVMIX-VMIX)>0) XOR (((LASTMDAC-MDAC)>0)));
289 2 8013 CALL PHASECALC;
290 2 8014 IF (( IABS(CALCDAC-LASTMDAC) > INT(4*512*DELTADAC))
291 3 8015 OR ( IABS(VMIX)> 7F00H ) ) /* out of range? */
292 3 8016 THEN DO;
293 3 8017 TPASS=0;
294 3 8018 LOCKED=FALSE;
295 3 8019 RETURN; /* lost LOCK....exit */
296 2 8020 END;
297 3 8021 IF ((SLOPE) XOR (LASTVMIX > 0)) THEN
298 3 8022 DO:
299 3 8023 IF (DAC+DELTADAC < DAC) THEN TPASS=0; /* inc ovf ? */
300 3 8024 DAC=DAC + DELTADAC ; /* inc dac */
301 2 8025 END;
302 3 8026 ELSE
303 3 8027 DO:
304 3 8028 IF (DAC-DELTADAC > DAC) THEN TPASS=2; /* dec ovf? */
305 3 8029 DAC=DAC - DELTADAC; /* dec dac */
306 2 8030 END;
307 2 8100 LASTVMIX=VMIX;
308 2 8150 LASTMDAC=MDAC;
309 2 8200 RETURN;
310 2 8250 END TRACKER;
8270 SEJECT

```



```

      8680 /*=====*/
      8690
338 1 8700 CTL_BLK_FTP:  PROCEDURE;
      8710
      8720 /* Start FTP */
      8730
339 2 8740     IF CTL_CSUM(4) OR FTP_ON THEN RETURN;
      8750
341 2 8760     FTP_ON = TRUE;
      8770
342 2 8780     FTP_SAMPLE_NO = 0;
343 2 8790     FTP_CHAN = LOW(CTL_BLK_LOC(1)) AND 0FH;
344 2 8800     FTP_DELAY = CTL_BLK_LOC(2) AND 01FFH;
345 2 8810     FTP_STEP = (CTL_BLK_LOC(3) AND 0007H);
      8820
346 2 8830     IF MK_2_PROGRAM THEN
347 2 8840         DO:
348 3 8850         HEAD_FTP_CHAN_LOC = LOW(FTP_CHAN); /*SEND OUT CHAN NUM*/
349 3 8860         FTP_CURRENT_DELAY = FTP_DELAY; /* SET UP DELAY*/
350 3 8870         ENDDO
      8880     ENDIF
      8890
351 2 8900     IF SB_PROGRAM THEN
352 2 8910         DO:
353 3 8920         HEAD_FTP_CHAN_LOC = LOW(FTP_CHAN);
354 3 8930         HEAD_FTP_DELAY_LOC = LOW(SHR(FTP_DELAY,1));
355 3 8940         IF (FTP_CHAN AND 10H) = 0
356 3 8950             THEN STA_ID_PDU_LOC = STA_ID_PDU XOR 0F8H;
357 2 8960             ELSE STA_ID_PDU_LOC = (STA_ID_PDU XOR 0F8H) AND 0F8H;
      8970         ENDIF
      8980         ENDDO
      8990     ENDIF
      9000
359 2 9010     FTP_MODE = 1;
      9020
360 2 9030     STATUS_FLG_LOC = 01H; /* DISABLE CAMAC READS FOR NOW*/
      9040
361 2 9050     RETURN;
362 2 9060 END CTL_BLK_FTP;
      9070
      9080 $ EJECT

```

```

310 1      8350 LOCKER:      PROCEDURE;
311 2      8360 IF (LPASS=2) THEN      /* first time locker entry ? */
312 2      8370 DO;
313 3          8380 DAC=0;      /* start with phase shift = 0 */
314 3          8390 LPASS=1;
315 3          8400 RETURN;
316 3      8410 END;
317 2      8420 IF (LPASS=1) THEN
318 2      8430 DO;
319 3          8440 IF (VMIX > 0) THEN      /* remember sign of the Vmix */
320 3              8450 PDAC0=FALSE;
321 3              8460 ELSE
322 3                  8470 PDAC0=TRUE;
323 3                  8480 LPASS=2;
324 3                  8490 DAC=80H;      /* phase shift = 90 degrees */
325 3                  8500 RETURN;
326 2      8510 END;
327 2      8520 IF (VMIX < 0) XOR (PDAC0) THEN      /* Vmix changed sign ? */
328 2          8530 DAC= DAC ~ SHR(80H,LPASS-2);      /* return to previous DAC */
329 2      8540 IF (LPASS=10) THEN      /* LOCKER routine done? */
330 2          8550 DO;
331 3              8560 LPASS=0;
332 3              8570 LOCKED=TRUE;
333 3              8580 RETURN;      /* exit */
334 2              8590 END;
335 2      8600 LPASS=LPASS+1;
336 2      8610 DAC= DAC ~ SHR(80H,LPASS-2);
337 2      8620 RETURN;
          8630 END LOCKER;
          8640
          8650
          8660
          8670 $ EJECT

```

```

          9090 /*=====*/
          9100
363      1      9110 CTL_BLK_SYS_V:  PROCEDURE;
          9120
          9130 /* Read in system voltages from remote device */
          9140
364      2      9150         IF CTL_CSUM(1) THEN RETURN;
          9160
366      2      9170         IF FTP_ON THEN RETURN; /* WE WONT INTERFERE */
          9180
368      2      9190         CALL SETW(0,6FTP_BLK_LOC,FTP_BLK_LEN);
          9200
369      2      9210         FTP_ARRAY(0) = 1000H;
          9220
370      2      9230         FTP_SAMPLE_NO = FTP_SAMPLES; /* SO BUFFER IS UPDATED */
          9240
371      2      9250         RETURN;
          9260
372      2      9270 END CTL_BLK_SYS_V;
          9280
          9290 $ EJECT
```

```
9300 /*=====*/
9310
373 1 9320 CTL_BLK_NEW:  PROCEDURE;
9330
9340 /* Bring in new parameters */
9350
374 2 9360 IF CTL_CSUM(12) THEN RETURN;
9370
376 2 9380 CLK_MINUTES = LOW(SHR(CTL_BLK_LOC(1),10));
377 2 9390 TRIG_RATE = HIGH(CTL_BLK_LOC(2));
378 2 9400 STA_ID_PDU = LOW(CTL_BLK_LOC(2));
379 2 9410 RETURN;
380 2 9420 END CTL_BLK_NEW;
9430
9440 * EJECT
```

```

9450 /*=====*/
381 1 9460 CTL_BLK_PRIMITIVES: PROCEDURE;
9470
9480 /* READS AND WRITES INTO MEMORY DIRECTLY */
9490
382 2 9500 IF CTL_BLK_LOC(0) = 6 THEN
383 2 9510 DO;
9520 IF CTL_CSUM(2) THEN RETURN; /*
384 3 9530 FTP_BLK_LOC(1) =
9540 DOUBLE(INPUT(CTL_BLK_LOC(1)));
385 3 9550 GOTO L1;
386 3 9560 ENDDO
9570 ENDDO
9580 ENDDO
9590
387 2 9600 IF CTL_BLK_LOC(0) = 7 THEN
388 2 9610 DO;
9620 IF CTL_CSUM(2) THEN RETURN; /*
389 3 9630 FTP_BLK_LOC(1) = INWORD(CTL_BLK_LOC(1));
390 3 9640 GOTO L1;
391 3 9650 ENDDO
9660 ENDDO
9670
392 2 9680 IF CTL_BLK_LOC(0) = 8 THEN
393 2 9690 DO;
9700 IF CTL_CSUM(3) THEN RETURN; /*
394 3 9710 OUTPUT(CTL_BLK_LOC(1)) = LOW(CTL_BLK_LOC(2));
395 3 9720 ENDDO
9730 ENDDO
9740
396 2 9750 IF CTL_BLK_LOC(0) = 9 THEN
397 2 9760 DO;
9770 IF CTL_CSUM(3) THEN RETURN; /*
398 3 9780 OUTWORD(CTL_BLK_LOC(1)) = CTL_BLK_LOC(2);
399 3 9790 ENDDO
9800 ENDDO
9810 /* I HAVE COMMENTED OUT THE HEAD ADC OPERATIONS HERE
9820 IF CTL_BLK_LOC(0) = 0AH THEN
9830 DO;
9840 IF CTL_CSUM(2) THEN RETURN;
9850 FTP_BLK_LOC(1) = UNSIGN(GETV(LOW(CTL_BLK_LOC(1))));
9860 GOTO L1;
9870 ENDDO
9880 ENDDO
9890 AND BACK TO THE CODE */
400 2 9900 RETURN;
9910
401 2 9920 L1:
9930 FTP_BLK_LOC(0) = 8000H;
402 2 9940 FTP_ON = FALSE;
403 2 9950 FTP_SAMPLE_NO = FTP_SAMPLES;
404 2 9960 STATUS_FLG_LOC = 05H;
405 2 9970 RETURN;
9980
406 2 9990 END CTL_BLK_PRIMITIVES;
10000
10010

```

```

10030 /*=====*/
10040
407 1 10050 GET#RAWSDATA: PROCEDURE INTERRUPT 100;
10060
408 2 10070 DECLARE (IB,JB,COUNT) BYTE.
10080 (IW,JW) WORD,
10090 (IL,JL) INTEGER,
10100 (TL,TL) LOGICAL;
10110
409 2 10120 CLK_PERIODS = CLK_PERIODS + 1;
410 2 10130 INTERRUPTED = TRUE;
411 2 10140 IB = TRIC_REG_LOC;
412 2 10150 TRIC_REG_LOC = 91B;
10160 /*=====*/
10170 /
10180 /
10190 /*=====*/
413 2 10200 CALL TRACE(7);
414 2 10210 CALL TIME(2); /* LET ADC DIGITIZE FOR 200 USEC */
415 2 10220 STATUS$REG;
416 2 10230 IF (STATUS AND FRESH$UPDATE$MASK)=0 THEN
417 2 10240 DO;
418 3 10250 RETURN;
419 3 10260 END;
420 2 10270 IF ((COMMAND$REG AND 08H) = 0) THEN B_OFFSET=0;
422 2 10280 ELSE B_OFFSET=1;
423 2 10290 ADC$PTR = CHEAD(B + 16*B_OFFSET);
424 2 10300 IF (WOBBLER = STATUS AND WOBBLER$MASK)=0 THEN
425 2 10310 RAWVMP=SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(PHASE$PTR)));
426 2 10320 ELSE
427 2 10330 RAWVMP=SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(PHASE$PTR)));
10340 VAMPLITUDE=SIGNED(SIGNFLIP XOR UNSIGN(
10350 ADC$DATA$BLOCK(AMP$PTR) - VAMP_OFFSET));
428 2 10360 MDAC= SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(DAC$PTR)));
429 2 10370 VZERO=SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(ZERO$PTR)));
430 2 10380 VTEMP=SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(TEMP$PTR)));
431 2 10390 VP15 =SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(P15$PTR)));
432 2 10400 VM15 =SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(P15$PTR)));
433 2 10410 V6AU6=SIGNED(SIGNFLIP XOR UNSIGN(ADC$DATA$BLOCK(V6AU6$PTR)));
434 2 10420 TEMP_NOW= VTEMP - VZERO ;
435 2 10430 IF TRACKING_MODE THEN DO;
437 3 10440 IF ((COUNT) THEN
438 3 10450 DO;
439 4 10460 VMIX=(RAWVNPT-RAWVMP); /* TEST FOR OVERFLOW NEEDS */
440 4 10470 IF SIGNED(UNSIGN(RAWVNPT) XOR UNSIGN(RAWVMP))=0 THEN GOTO OKRANGE;
442 4 10480 IF (RAWVMP>0 AND VMIX<0) THEN VMIX=PLUS$FULL$SCALE;
444 4 10490 IF (RAWVMP<0 AND VMIX>0) THEN VMIX=MINUS$FULL$SCALE;
446 4 10500 OKRANGE: VMPED= RAWVMP/2 + RAWVMP/2 ;
447 4 10510 AVE_VAMP= VAMPLITUDE/2 + AVE_VAMP/2;
448 4 10520 AMP_JITTER= VAMPLITUDE - AVE_VAMP ;
449 4 10530 AVE_AMP_JITTER = IABS(AMP_JITTER)/2 + AVE_AMP_JITTER/2;
450 4 10540 AVE_VMPED= VMPED/2 + AVE_VMPED/2;
451 4 10550 IF (LOCKED) THEN DO;
453 5 10560 CALL TRACKER;
454 5 10570 CALL INDICATOR(GREEN_LIGHT);
455 5 10580 END;
456 4 10590 IF NOT(LOCKED) THEN DO;

```

```

458 5      10600      CALL LOCKER;
459 5      10610      CALL INDICATOR(REF_LIGHT);
460 5      10620      END;
461 4
462 3      10640      END;
          10650      COUNTR=COUNTR +1;      /* NEEDS MORE THOUGHT HERE*/
          10660      /* CALL UPDATE_STATUS_BLK ; */
          10670      IF (GET_AMP_PED) THEN DO:
463 3      10680      COMMAND$REG=(COMMAND$REG OR 80H);      /* OR IN DIGITIZE BIT*/
465 4      10690      COMMAND$REG=(COMMAND$REG AND 7FH);      /* LET IT GO */
466 4      10700      CALL TIME (2);      /* LET ADC DIGITIZE */
467 4      10710      IF ((COMMAND$REG AND 08) = 0) THEN B_OFFSET=0;
468 4      10720      ELSE B_OFFSET=1;
469 4      10730      ADC$POINTER= 0HEAD(8 + 16*B_OFFSET);
470 4      10740      VAMP_OFFSET= ADC$DATA$BLOCK(AMP$PTR);
471 4      10750      VAMP_OFF_OUT=SIGNED(SIGNFLIP XOR UNSIGN(VAMP_OFFSET));
472 4      10760      GET_AMP_PED=FALSE;
473 4      10770      COMMAND$REG=(COMMAND$REG OR 80H);      /* OR IN DIGITIZE BIT*/
474 4      10780      COMMAND$REG=(COMMAND$REG AND 7FH);      /* LET IT GO */
475 4      10790      END;
476 3      10800      END;
477 2      10810      RETURN;
478 2      10820      END GET$RAW$DATA;
479 2      10830
480 2      10840      * EJECT

```

```
10850 /*=====*/
10860
481 1 10870 HEAD_INTERRUPT PROCEDURE INTERRUPT 11H;
10880
482 2 10890 DECLARE (1B.JB) BYTE,
10900 (1W.JW) WORD,
10910 (1I.JI) INTEGER,
10920 (1L.JL) LOGICAL;
10930
483 2 10940 INTERRUPTED = TRUE;
10950 /* DISABLE INTERRUPT SOURCE */
484 2 10960 RESET_NMI_LOC = 0H;
485 2 10970 RETURN;
486 2 10980 ;
487 2 10990 END HEAD_INTERRUPT;
11000
11010
11020 * EJECT
```



```

11030 /*=====*/
488 1 11040 SERVICE_CAMAC: PROCEDURE;
11050
11060 /* Unloads any new control block of data from CAMAC */
11070
11080 /* if control block is not full, then return */
489 2 11090 IF (STATUS_FLG_LOC AND 01H) = 01H THEN RETURN;
11100
491 2 11110 IF (CTL_BLK_LOC(0) > 00H) AND (CTL_BLK_LOC(0) <= 05H) THEN
492 2 11120 DO;
493 3 11130 DO CASE (CTL_BLK_LOC(0) - 1);
11140
494 4 11150 CALL CTL_BLK_FTP;
11160
495 4 11170 CALL CTL_BLK_SYS_V;
11180
496 4 11190 GOTO L3;
11200
497 4 11210 GOTO L3;
11220
498 4 11230 GOTO L3;
11240
499 4 11250 ENDDO
500 3 11260 GOTO L1;
501 3 11270 ENDDO
11280
11290 ENDF
502 2 11300 IF (CTL_BLK_LOC(0) >= 06H) AND (CTL_BLK_LOC(0) <= 0AH) THEN
503 2 11310 DO;
504 3 11320 CALL CTL_BLK_PRIMITIVES;
505 3 11330 GOTO L1;
506 3 11340 ENDDO
11350 ENDF
11360
11370 /*
11380 I HAVE COMMENTED OUT KEITH'S COMMAND BLOCK
11390 CODING BUT LEFT IT AS A TEMPLATE FOR FUTURE
11400 THOUGHT
11410
11420
11430 IF MK_2_PROGRAM AND (CTL_BLK_LOC(0) >= 10H) AND
11440 (CTL_BLK_LOC(0) <= 1AH) THEN
11450 DO;
11460 DO CASE (CTL_BLK_LOC(0) - 10H);
11470
11480 IF NOT CTL_CSUM(1) THEN
11490 CALL KLY_ON(1);
11500 ENDF
11510
11520 IF NOT CTL_CSUM(1) THEN
11530 CALL KLY_OFF(1);
11540 ENDF
11550
11560
11570 IF (NOT CTL_CSUM(1)) AND
11580 (FAULT_CNT_LIMIT < (255 - FAULT_TH)) THEN
11590 DO;

```

```

11600          FAULT_CNT_LIMIT = FAULT_COUNT + FAULT_TH;
11610          GOTO L1;
11620          ENDDO
11630      ELSE
11640          FAULT_COUNT = FAULT_CNT_LIMIT - FAULT_TH;
11650      ENDIF
11660
11670      IF (NOT CTL_CSUM(1)) AND
11680          (MOVE_PHASE_S = 0) THEN
11690          MOVE_PHASE_S = 1;
11700      ENDIF
11710
11720      CALL CTL_NEW_PHASE;
11730
11740      CALL CTL_DELTA_PHASE;
11750
11760      IF NOT CTL_CSUM(2) THEN
11770          DRIVE_REQ = LOW(CTL_BLK_LOC(1));
11780      ENDIF
11790
11800      GOTO L3;
11810
11820      IF NOT CTL_CSUM(1) THEN
11830          DO;
11840              CALL KLY_STOP(8.0,DRIVE_REQ);
11850              CALL KLY_ON(2);
11860          ENDDO
11870      ENDIF
11880
11890      GOTO L3;
11900
11910      GOTO L3;
11920
11930      !
11940      AND WE RETURN TO REAL CODE RIGHT HERE
11950      IF PHASCOD AND (CTL_BLK_LOC(0) = 102B) THEN */
11960      DO;
11970          IF CTL_CSUM(6) THEN RETURN;
11980          TRACKING_MODE=TRUE;
11990          STA_ID_PDU=LOW(CTL_BLK_LOC(1)) AND 07H;
12000          STA_ID_PDU_LOC= STA_ID_PDU OR
12010              SHL((NOT STA_ID_PDU).3);
12020          WOBBLESCTRLSDIVISOR=LOW(CTL_BLK_LOC(2));
12030          CLOCKSCTRLSDIVISOR= LOW(CTL_BLK_LOC(3));
12040          DELTADAC=LOW(CTL_BLK_LOC(4));
12050          /* FUTURE OPTIONS AT CTL_BLK_LOC(5) */
12060      END;
12070      IF PHASCOD AND (CTL_BLK_LOC(0) = 101B) THEN
12080      DO;
12090          TRACKING_MODE=FALSE;
12100      END;
12110      IF PHASCOD AND (CTL_BLK_LOC(0) = 011CH) THEN
12120      DO;
12130          CALL CTL_BLK_NEW;
12140
12150          ENDDO
12160          GOTO L1;

```

```

507 2
508 2
509 3
511 3
512 3
513 3
514 3
515 3
516 3
517 3
518 2
519 2
520 3
521 3
522 2
523 2
524 3
525 3
526 2

```

```
12170
12180 /* WE HAVE NOT FOUND A ROUTINE YET */
12190
527 2 12200 L3:    CALL ERROR_SEND(SYS_ERR_CTL_BAD);
      12210
      12220
528 2 12230 L1:    STATUS_FLG_LOC = 04H; /* SET CONTROL BLOCK */
529 2 12240      RETURN;
      12250
530 2 12260 END SERVICE_CAMAC;
      12270
      12280 $ EJECT
```

```

12290 /*=====*/
12300
12310
12320 /*=====*/
12330 /*=====*/
12340 /*=====*/
12350
531 1 12360 PROGRAM_CODE;
12370 COLDSTART;
12380 DISABLE;
12390
532 1 12400 CALL SET%INTERRUPT(2,NMI_LINK_INT);/* INITIALIZE NMI */
533 1 12410 CALL SET%INTERRUPT(18H,GET%RAW%DATA); /* IDLE TRIGGER */
534 1 12420 CALL SET%INTERRUPT(11H,HEAD_INTERRUPT);
535 1 12430 CALL SET%INTERRUPT(19H,GET%RAW%DATA); /*COLLISION DESTINATION*/
536 1 12440 STA_ID_PDU = 0;
537 1 12450 STA_ID_PDU_LOC = STA_ID_PDU; /* SET FRONT PANEL LIGHTS ON */
12460
538 1 12470 CALL SETW (0000H,@STATUS_BLK(0),STATUS_BLK_LEN);
12480 /* SET STATUS BLOCK TO 0 */
12490
12500 /* Set up DMA channel requirements */
539 1 12510 CALL DMA_INIT;
540 1 12520 CALL HEAD_RESET;
541 1 12530 CALL INDICATOR(NO_LIGHT);
542 1 12540 SOFT_REL_NUMBER = SOFT_REL;
12550
543 1 12560 FTP_ON = FALSE;
544 1 12570 FTP_MODE = 0;
545 1 12580 FTP_SAMPLE_NO = 0;
12590
16 1 12600 GOOD_TRIES = 0; /* STARTING VECTORS FOR THE */
547 1 12610 BAD_CABLE = TRUE; /* CABLE CHECKING ROUTINE */
12620
548 1 12630 CLK_MINUTES = 0;
549 1 12640 CLK_PERIODS = 0;
550 1 12650 LPASS=0;
551 1 12660 TPASS=0;
552 1 12670 STATUS_FLG_LOC = 04H; /* ENABLE CTL_BLOCK */
12680
12690 /* Clear interrupt type 18H on power up. */
553 1 12700 1B = TRIG_REC_LOC;
554 1 12710 1B=RESET_NMI_LOC;
555 1 12720 LOCKED=FALSE;
556 1 12730 COMMANDSREG= 13H;
557 1 12740 CLOCKS%CONTROLS%DIVISOR=1FH;
558 1 12750 WOBBLER%CONTROLS%DIVISOR=00;
559 1 12760 DELTADAC= 01 ; /****/
560 1 12770 GET_AMP_PED=TRUE;
561 1 12780 TRACKING_MODE=TRUE;
562 1 12790 TRIG_REC_LOC= 91H; /* EKABLE IDLE TRIG */
563 1 12800 ENABLE; /* INTERRUPTS */
12810
12820 /* To initialize some STD_HEAD Parameters:
12830
12840 GOTO BACKGROUND;
12850

```

```

12860 /*=====*/
12870
564 1 12880 BACKGROUND: /* NOTE: GETSRWDATA IS CALLED VIA INTERRUPT */
12890
12900 CALL CLOCK;
565 1 12910 CALL TRACE(1);
12920 /* CALL CHECK_CABLE; */
566 1 12930 CALL TRACE(2);
567 1 12940 CALL SERVICE_CAMAC; /* Accept new control block */
568 1 12950 CALL TRACE(3);
12960
12970
12980
569 1 12990 CALL UPDATE_FTP;
570 1 13000 CALL TRACE(4);
571 1 13010 CALL UPDATE_STATUS_WORD;
572 1 13020 CALL TRACE(5);
573 1 13030 IF CLK_PERIODS < OLD_CLK_PERIODS THEN
574 1 13040 DO;
575 2 13050 OLD_CLK_PERIODS = CLK_PERIODS;
576 2 13060 ENDDO
577 1 13070 CALL TRACE(6);
578 1 13080 CALL UPDATE_STATUS_BLK;
13090
13100
13110
13120
13130
13140
13150
579 1 13160 /*=====*/
13170 /*=====*/
13180 /* I THINK THIS IS THE END */
13190 END;

```

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES
19	0000H	16	ADCDATABLOCK INTEGER BASED(ADCP0INTER) ARRAY(8) 425 426
17	00E2H	1	ADCOFFSET. BYTE 430 431 432 433 472
18	0028H	4	ADCP0INTER POINTER 423* 425 426 427 428 429 430
38	005EH	2	ADC_0. INTEGER AT 433 471* 472
38	0060H	2	ADC_1. INTEGER AT
23			ANPPTR LITERALLY '2' 427 472
40	005EH	2	AMP_JITTER INTEGER AT 448* 449
40	005AH	2	AVE_AMP_JITTER INTEGER AT 449 449*
39	003AH	2	AVE_CALCPHASE. INTEGER AT 271 272 272*
40	0056H	2	AVE_PHASE_JITTER INTEGER AT 273 273*
39	003EH	2	AVE_VAMP INTEGER AT 447 447* 440
39	0042H	2	AVE_VMPED. INTEGER AT 450 450*
564	019AH		BACKGROUND LABEL 579
161	0086H	2	BADTRIES WORD IN PROC (UPDATE_STATUS_BLK) 174 174*
161			BADTRIESLIMIT. LITERALLY '4000' IN PROC (UPDATE_STATUS_BLK)
8	00C8H	1	BAD_CABLE. BYTE 109 112* 268 220 237 239* 247* 54*
8	00C9H	1	BAD_CABLE_TEST BYTE 107* 219* 227 234
38	0044H	2	BEAM_CUR INTEGER AT
38	0046H	2	BEAM_CUR_TR. INTEGER AT
38	0040H	2	BEAM_VOLTS INTEGER AT
38	0042H	2	BEAM_VOLTS_TR. INTEGER AT
134	0414H	43	BLINK. PROCEDURE STACK=0002H
61	0000H	2	BLOCK. WORD BASED(BLOCKPTR) ARRAY(1) IN PROC (CHECK_SUM)
161	00E7H	1	BLOCK2 BYTE IN PROC (UPDATE_STATUS_BLK) 163 170
60	0006H	4	BLOCKPTR POINTER IN PROC (CHECK_SUM) PARAMETER AUTOMATIC
10	00DFB	1	B_OFFSET BYTE 421* 422* 4 4 469* 470* 471
13	0024H	2	CALCDAC. INTEGER 266* 267* 269 270 289
39	0038H	2	CALCPHASE. INTEGER AT 269* 270* 271 272
217	05C5H	140	CHECK_CABLE. PROCEDURE STACK=000CH
59	0225H	76	CHECK_SUM. PROCEDURE BYTE STACK=0008H 73
8	00CDH	1	CLK_MINUTES BYTE 122 122* 124 126 126* 130 376* 54*
8	0000H	2	CLK_PERIODS. WORD 117 120 120* 143 151 409 409* 549
8			CLK_PERIODS_MIN. LITERALLY '60*180' 117 120
115	03CEH	70	CLOCK. PROCEDURE STACK=0002H 564
20	4005H	1	CLOCKCONTROLDIVISOR. BYTE AT ABSOLUTE 515* 537*
531	08ADH		COLDSTART. LABEL
20	4002H	1	COMMANDREG BYTE AT ABSOLUTE 259 261* 420 465 465* 4
408	00EDH	1	COUNT. 475 473* 476 476* 556*
338	08A5H	105	CTL_BLK_FTP. BYTE IN PROC (GETRANDATA) 437 462 462*
32			CTL_BLK_LEN. PROCEDURE STACK=0012H 494
36	5080H	32	CTL_BLK_LOC. LITERALLY '14' CB 89
			WORD ARRAY(1.) AT ABSOLUTE 73 343 344 3
373	0945H	57	CTL_BLK_NEW. 378 382 384 387 389 392 394 396 398 491
301	097EH	121	CTL_BLK_PRIMITIVES. 502 507 512 514 515 516 518 522
363	090EH	55	CTL_BLK_SYS_V. PROCEDURE STACK=0012H 524
71	0271H	37	CTL_CSUN PROCEDURE STACK=0012H 504
20	4004H	1	DAC. PROCEDURE BYTE STACK=000EH 495
			PROCEDURE STACK=000EH 359 364 374 5
			BYTE AT ABSOLUTE 262 282* 283 283* 297 2
			304 304* 313* 323* 327 327* 333 333*

24		DACPTR	LITERALLY '4'	428	
38	0062H	2 DAC_S	WORD AT		
44	0004H	1 DELTA	BYTE IN PROC (TRACE) PARAMETER AUTOMATIC	44	4
10	00E0H	1 DELTADAC	BYTE	282 283 289 297 299 302 304 316*	53
78	0296H	247 DMA_INIT	PROCEDURE STACK=0002H	539	
35	5500H	16 DMA_LOC	BYTE ARRAY(16) AT ABSOLUTE	79* 80* 81*	
			83* 86* 87* 88* 89* 90* 91* 92* 93*		
			96* 97* 98* 99* 100* 101* 102* 103* 173*		
			181* 182 184* 185* 188* 189* 190* 191*		
		DOUBLE	BUILTIN	130 384	
38	004AH	2 DRIVE	WORD AT		
38	0048H	2 DRIVE_REQ	WORD AT		
38	0056H	2 EM_CUR	INTEGER AT		
38	0058H	2 EM_CUR_TH	INTEGER AT		
4		ENDDO	LITERALLY 'END:'	54 127 147 186 228 2	
			249 349 357 385 390 394 395 500 507 524		
			575		
4		ENDIF	LITERALLY ' '	46 55 71 120 129 144 1	
			165 172 179 180 187 228 229 236 238 249		
			250 350 357 358 386 391 395 501 506 578		
50	0004H	2 ERRORWORD	WORD IN PROC (ERROR_SEND) PARAMETER AUTOMATIC	10	
49	01FAB	43 ERROR_SEND	PROCEDURE REENTRANT STACK=0004H	75 111 214	
8	00CBH	1 ERROR_SENT	BYTE	56*	
35	5810H	2 ERROR_WORD_LOC	WORD AT ABSOLUTE	53*	
4		FALSE	LITERALLY '000H'	69 74 146 162 201 209 2	
			223 247 292 320 346 351 403 414 520 543 555		
38	0054H	1 FAULT_COUNT	BYTE AT		
38	0055H	1 FAULT_COUNT_OLD	BYTE AT		
38	0030H	2 FE_POWER	INTEGER AT		
38	003AB	2 FE_POWER_TH	INTEGER AT		
12	003FH	1 FRESHUPDATE/MY	BYTE DATA	416	
36	50A6H	128 FTP_ARRAY	INTEGER ARRAY(64) AT ABSOLUTE	369*	
32		FTP_BLK_LEN	LITERALLY '(FTP_SAMPLES-3)'	93 94 197	
36	50A0H	134 FTP_BLK_LOC	WORD ARRAY(67) AT ABSOLUTE	36 168 384* 360	
36	50A0H	2 FTP_CHAN	WORD AT ABSOLUTE	343* 348 353 355	
36	006EH	2 FTP_CURRENT_DELAY	WORD	200* 349*	
36	006CH	2 FTP_DATA	INTEGER		
36	50A2H	2 FTP_DELAY	WORD AT ABSOLUTE	349 354	
36	00E5H	1 FTP_MODE	BYTE	199* 359* 36*	
36	00E4H	1 FTP_ON	BYTE	201* 339 341* 366 402* 543*	
36	0072H	64 FTP_PEDISTAL	INTEGER ARRAY(32)		
36	00D6H	1 FTP_PED_CNT	BYTE		
36		FTP_PED_DELAY	LITERALLY '60'	200	
36	0070H	2 FTP_PED_VALUE	INTEGER		
36		FTP_SAMPLES	LITERALLY '64'	93 94 197 368 370 403	
36	00E3H	1 FTP_SAMPLE_NO	BYTE	197 202* 342* 370* 403* 545*	
36	50A4H	2 FTP_STEP	WORD AT ABSOLUTE	343*	
407	09F7H	638 CETRA*DATA	PROCEDURE INTERF AT(24) STACK=000EH	533 535	
9	00D9H	1 GET_AHP_PED	BYTE	123* 463 47* 500*	
C	00CAH	1 GOOD_TRIES	BYTE	220 221* 24 245 245* 546*	
30		GREEN_LIGHT	LITERALLY '2'	4 4	
38	0052H	1 H2O_FAULT_CNT	BYTE AT		
38	0053H	1 H2O_FAULT_CNT_OLD	BYTE AT		
13	4020H	256 HEAD	INTEGER ARRAY(128) AT ABSOLUTE	28 233 233 471	
20	4001H	1 HEADID	BYTE AT ABSOLUTE		
20	40C0H	1 HEADTYPE	BYTE AT ABSOLUTE		

[illegible]


```

5          TEST_MODE. . . . . LITERALLY 'TRUE' 143 149
34 0030B 2 TIME . . . . . BUILTIN 137 414 467
38 0038B 6 TIME_STAMP. . . . . WORD AT 130*
10 000AB 6 TIM_OUT_INTERVALS. . . . . BYTE ARRAY(6) DATA
43 01DBB 31 TPASS. . . . . BYTE 277 281 284* 291* 298* 303* 551*
5 TRACE. . . . . PROCEDURE STACK=0004H 418 545 566 568 570 5
276 0717B 21B TRACE_MODE. . . . . LITERALLY 'TRUE' 45
9 0000B 1 TRACKER. . . . . PROCEDURE STACK=000AH 453
34 0034B 1 TRACKING_MODE. . . . . BYTE 435 511* 520* 561*
35 58.6B 1 TRIG_RATE. . . . . BYTE AT 377*
4 TRUE. . . . . BYTE AT ABSOLUTE 411 412* 553 562*
UNSIGN . . . . . LITERALLY 'OFFB' 45 56 68 76 79 85 10
196 0563B 40 UPDATE_FTP. . . . . BUILTIN 270 425 426 427 428 429 430 431
160 047BB 232 UPDATE_STATUS_BLK. . . . . 433 440 473
141 043FB 60 UPDATE_STATUS_WORD. . . . . PROCEDURE STACK=0002H 569
40 0064B 2 V6AU6. . . . . PROCEDURE STACK=0002H 578
28 V6AU6PTR. . . . . PROCEDURE STACK=0002H 571
39 003CH 2 VAMPLITUDE. . . . . INTEGER AT 433*
13 0026H 2 VAMP_OFFSET. . . . . LITERALLY '7' 433
40 005CB 2 VAMP_OFF_OUT. . . . . INTEGER AT 427* 447 448
3 0004B 52 VAX_FILE_NAME. . . . . INTEGER 427 472* 473
40 0062B 2 VM15. . . . . INTEGER AT 473*
39 0040R 2 VM1X. . . . . BYTE ARRAY(52) DATA
39 0040B 2 VNPED. . . . . INTEGER AT 432*
40 0060H 2 VP15. . . . . INTEGER AT 265 267 279 287 289 306 319 326
40 0063H 2 VTEMP. . . . . 442 443* 444 445*
40 0066H 2 VZERO. . . . . INTEGER AT 446* 450
60 0004H 1 WCOUNT. . . . . INTEGER AT 431*
20 4006H 1 WOBLECONTROLDIVISOR. . . . . INTEGER AT 430* 434
10 000BH 1 WOBBLER. . . . . INTEGER AT 429* 434
11 003EB 1 WOBBLERMASK. . . . . BYTE IN PROC (CHECK_SUM) PARAMETER AUTOMATIC
21 ZEROPTR. . . . . BYTE AT ABSOLUTE 514* 538*
MODULE INFORMATION:
CODE AREA SIZE = 0CF7B 3319D
CONSTANT AREA SIZE = 00A8B 16BD
VARIABLE AREA SIZE = 00F4B 244D
MAXIMUM STACK SIZE = 0042B 66D
1333 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS
DICTIONARY SUMMARY:
1000KB MEMORY AVAILABLE
4KB MEMORY USED (0%)
0KB DISK SPACE USED

```