NASA-CR-201536

SOLAR FLUX MONITOR

NAG5-1691

FINAL REPORT -- July 3, 1996

63985

SOLAR FLUX MONITOR

NAG5-1691

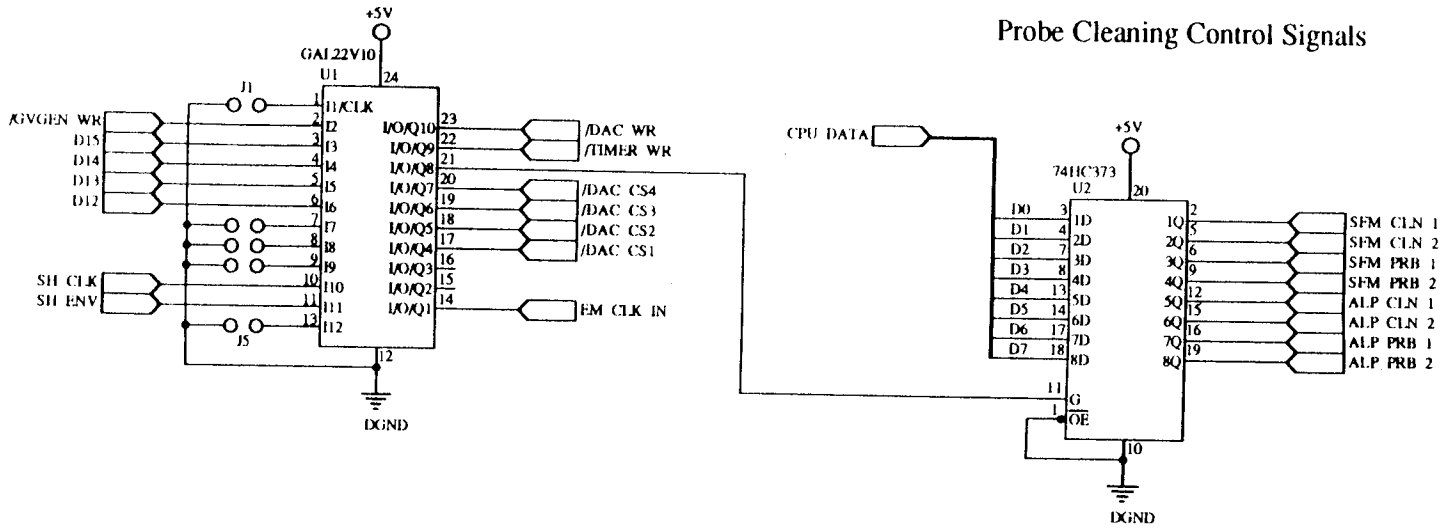
Final Report

July 3, 1996

George R. Carignan
Space Physics Research Laboratory
University of Michigan
2455 Hayward St.
Ann Arbor MI 48109-2143

S-F MON DWG.

Probe Cleaning Control Signals



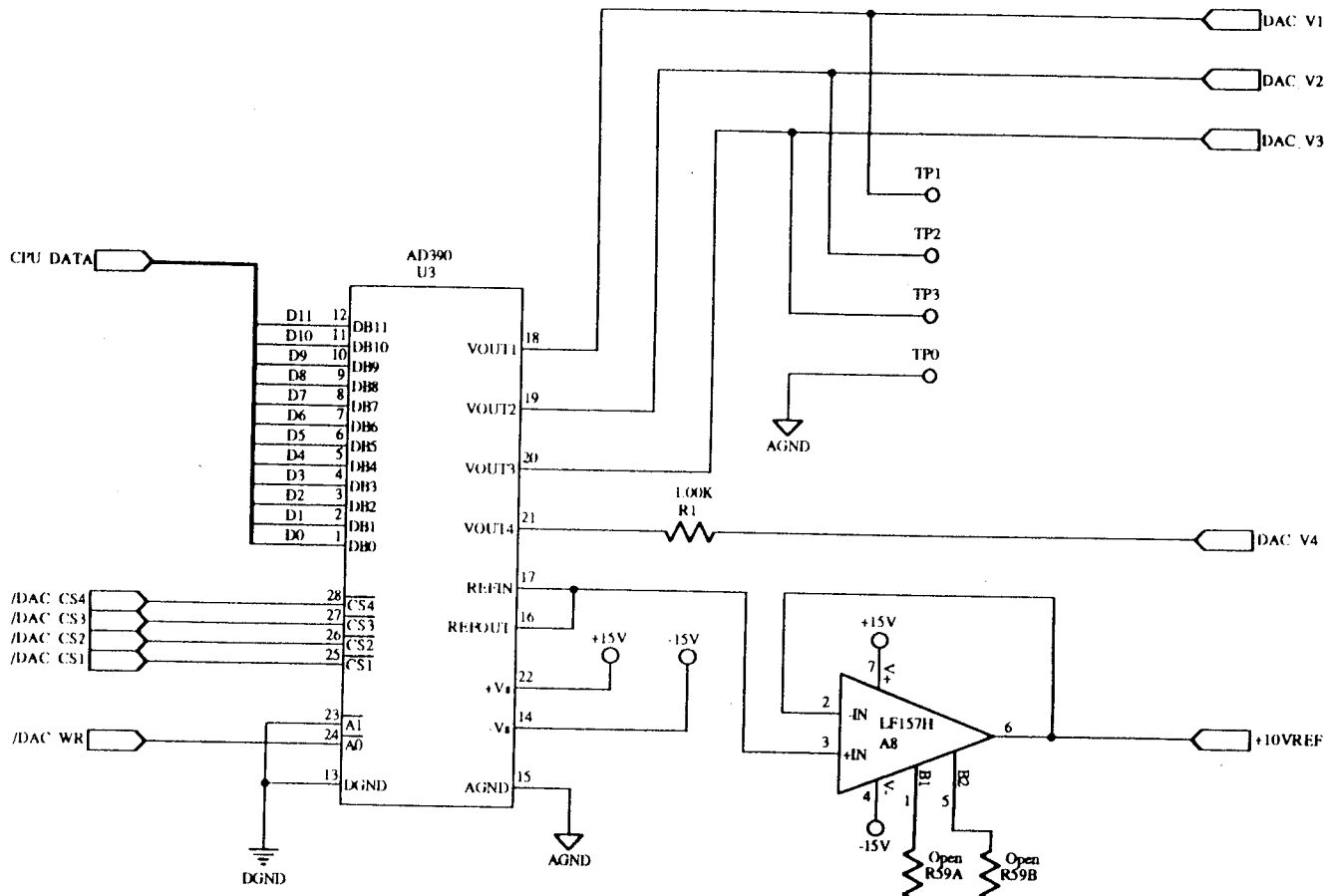
To write to:	Data must be of the form
Programmable Timer	00XXXXAADDDDDDDD
Quadruple DAC	01AADDDDDDDDDDDD
Octal Latch	10XXXXXXDDDDDDDD

NOTES:
1. Unless otherwise specified, all resistors are type RNC55 and all capacitors are type CKR05.

Space Physics Research Laboratory College of Engineering University of Michigan	Decoding PAL & Latch Grid Voltage Generator Solar Flux Monitor	REV REV CONTROL RELEASE DRAWN Dec. 31,1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

McCAD 3.2

1.1



D/A Converter

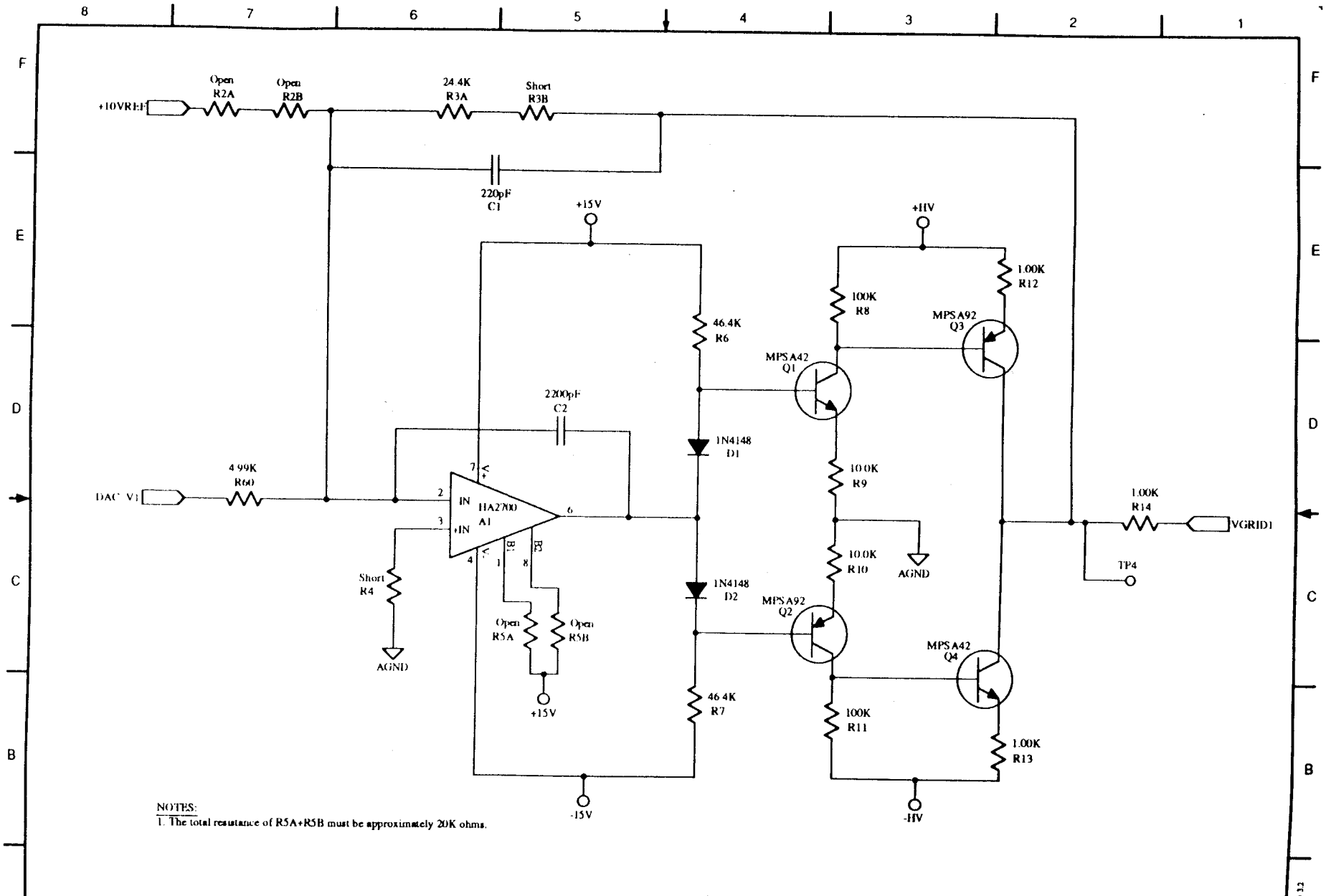
To write to:	Data must be of the form
Programmable Timer	00XXXXAADDDDDDDD
Quadruple DAC	01AADDDDDDDDDDDD
Octal Latch	10XXXXXXDDDDDDDD

NOTES:
1. The REFOUT signal (pin16) of U3 can typically tolerate 3.5mA for external use.

Space Physics Research Laboratory College of Engineering University of Michigan	D/A Converter Grid Voltage Generator Solar Flux Monitor	REV	Nov 14, 1992
		REV	
		CONTRL'D	
		RELEASE	
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER	DRAWN Sept 15, 1991
			ENGINEER N. Voronka

McCAD 3.2

1.2

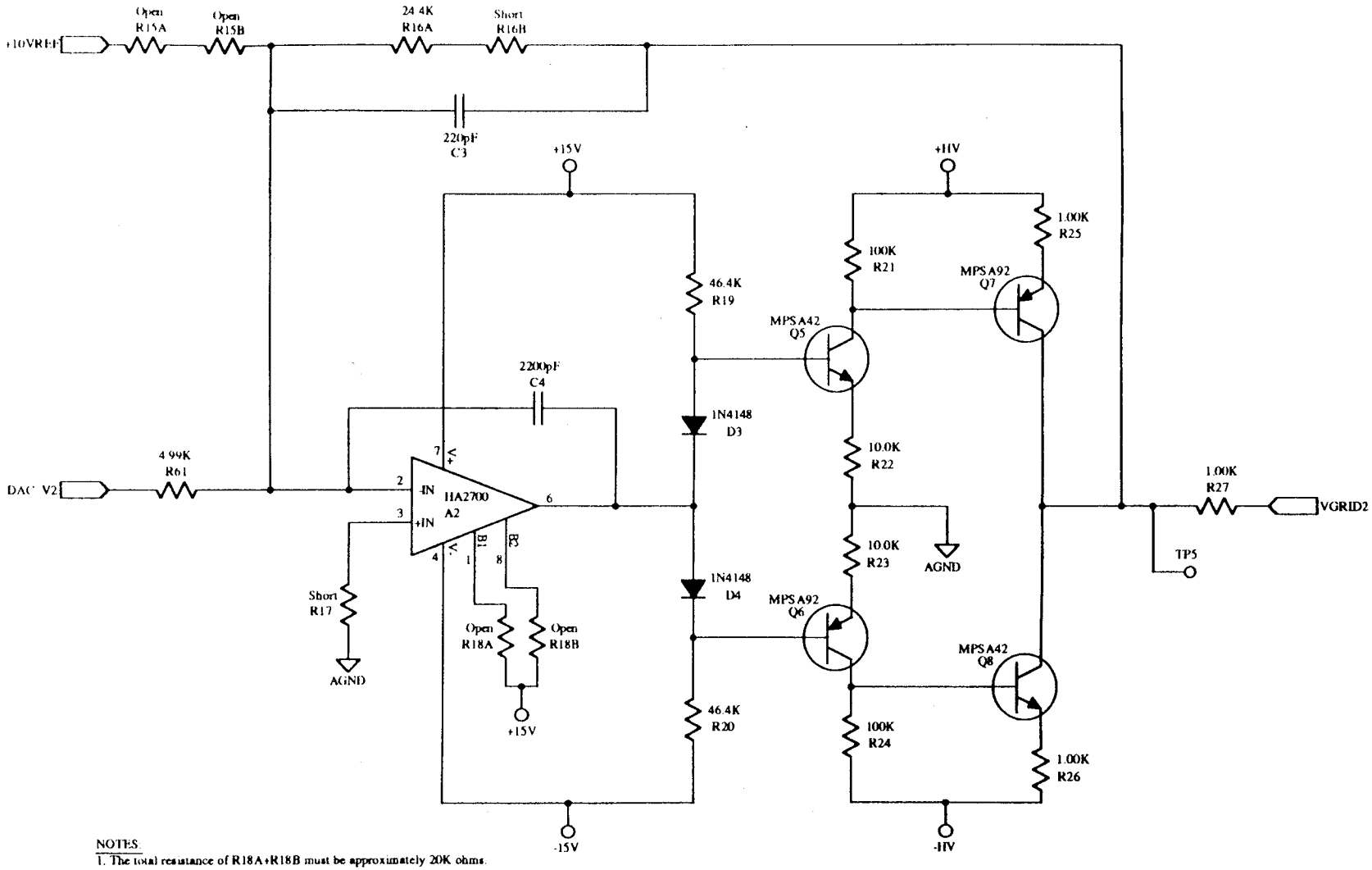


NOTES:
1. The total resistance of R5A+R5B must be approximately 20K ohms.

Space Physics Research Laboratory College of Engineering University of Michigan	Grid 1 Bipolar Amplifier Grid Voltage Generator Solar Flux Monitor	REV REV CONTRL'D RELEASE	Nov 14, 1992
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER	DRAWN Sept 15,1991 ENGINEER Block/Voronka

McCAD 3.2

1.3

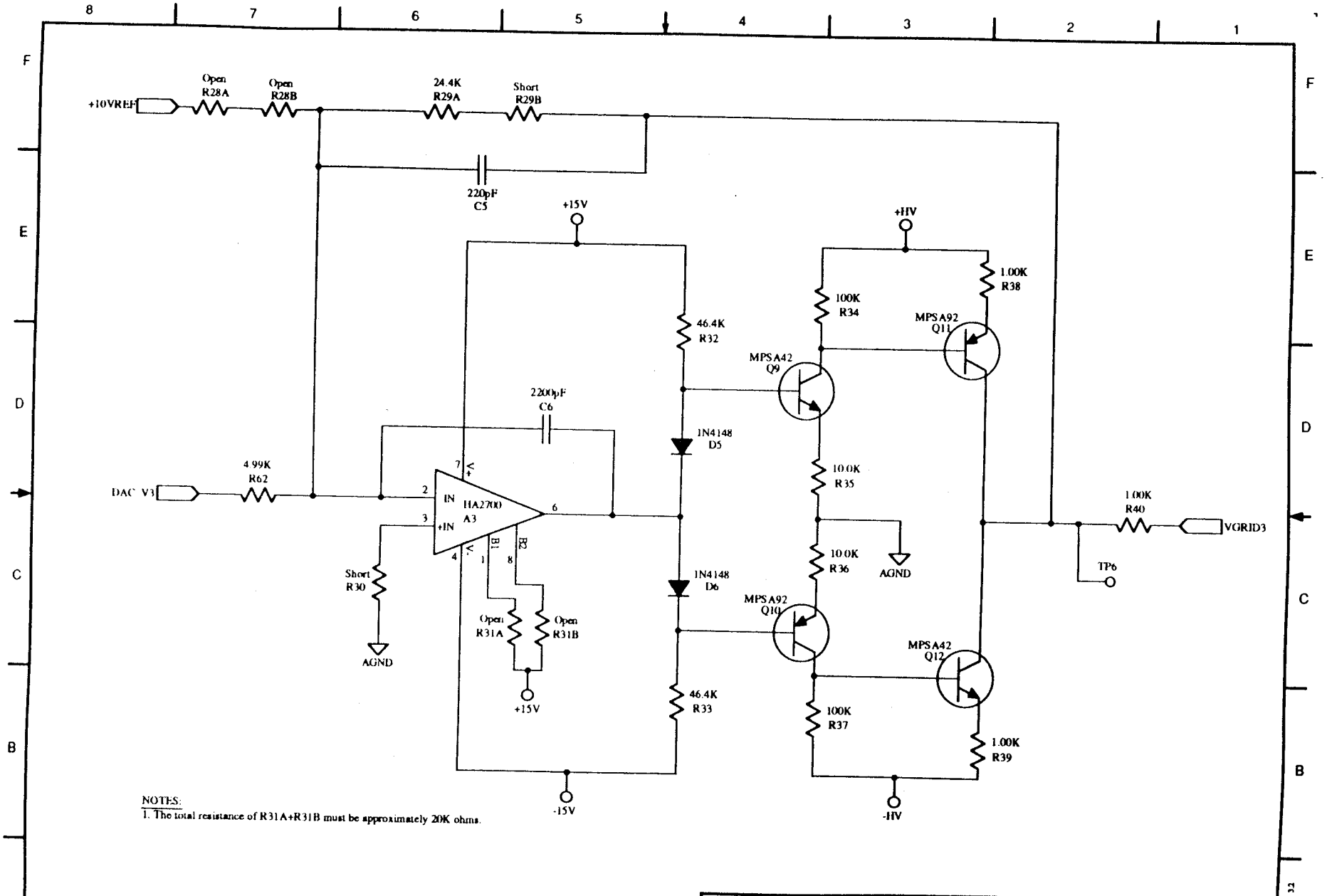


NOTES:
1. The total resistance of R18A+R18B must be approximately 20K ohms.

Space Physics Research Laboratory College of Engineering University of Michigan	Grid 2 Bipolar Amplifier Grid Voltage Generator Solar Flux Monitor	REV	Nov 14, 1992
		REV	
		CONTRLT	
		RELEASE	
PROJECT NAME: SFM/ALP	PILE NAME	DRAWN	Sept 15,1991
	DRAWING NUMBER	ENGINEER	Block/Voronka

MCAD 32

1.4

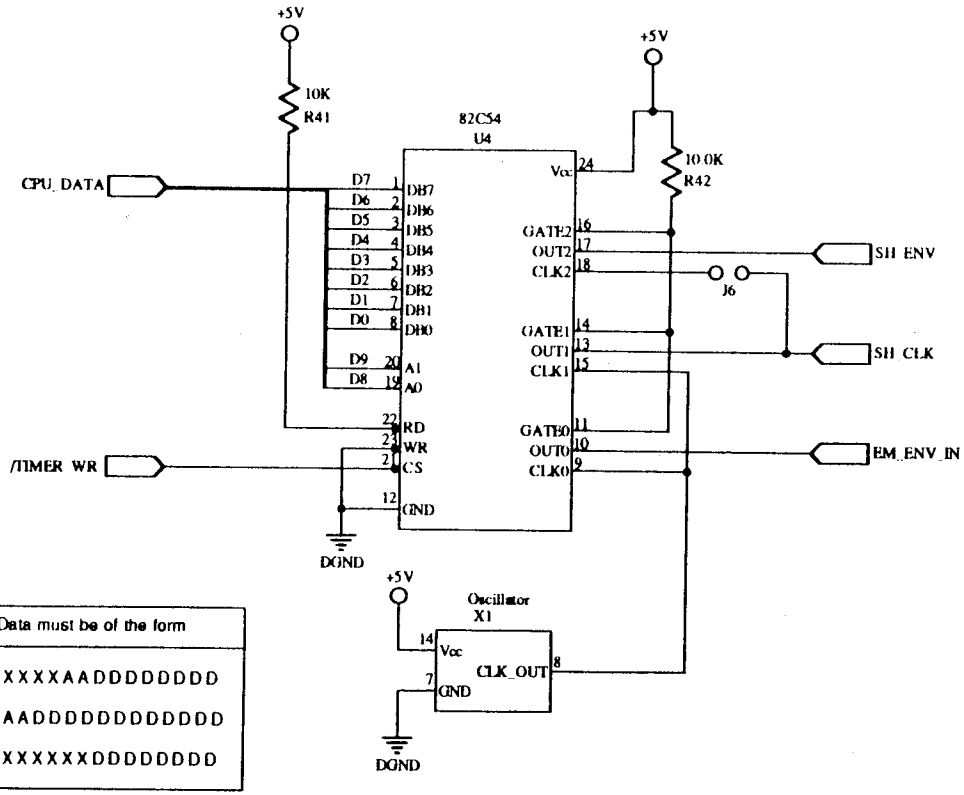


NOTES:
1. The total resistance of R31A+R31B must be approximately 20K ohms.

Space Physics Research Laboratory College of Engineering University of Michigan	Grid 3 Bipolar Amplifier Grid Voltage Generator Solar Flux Monitor	REV	Nov 14, 1992
		REV	
		CONTRL'D	
		RELEASE	
PROJECT NAME SFM/ALP	FILE NAME	DRAWN	Sept 15, 1991
	DRAWING NUMBER	ENGINEER	Block/Voronka

McCAD 3.2

1.5

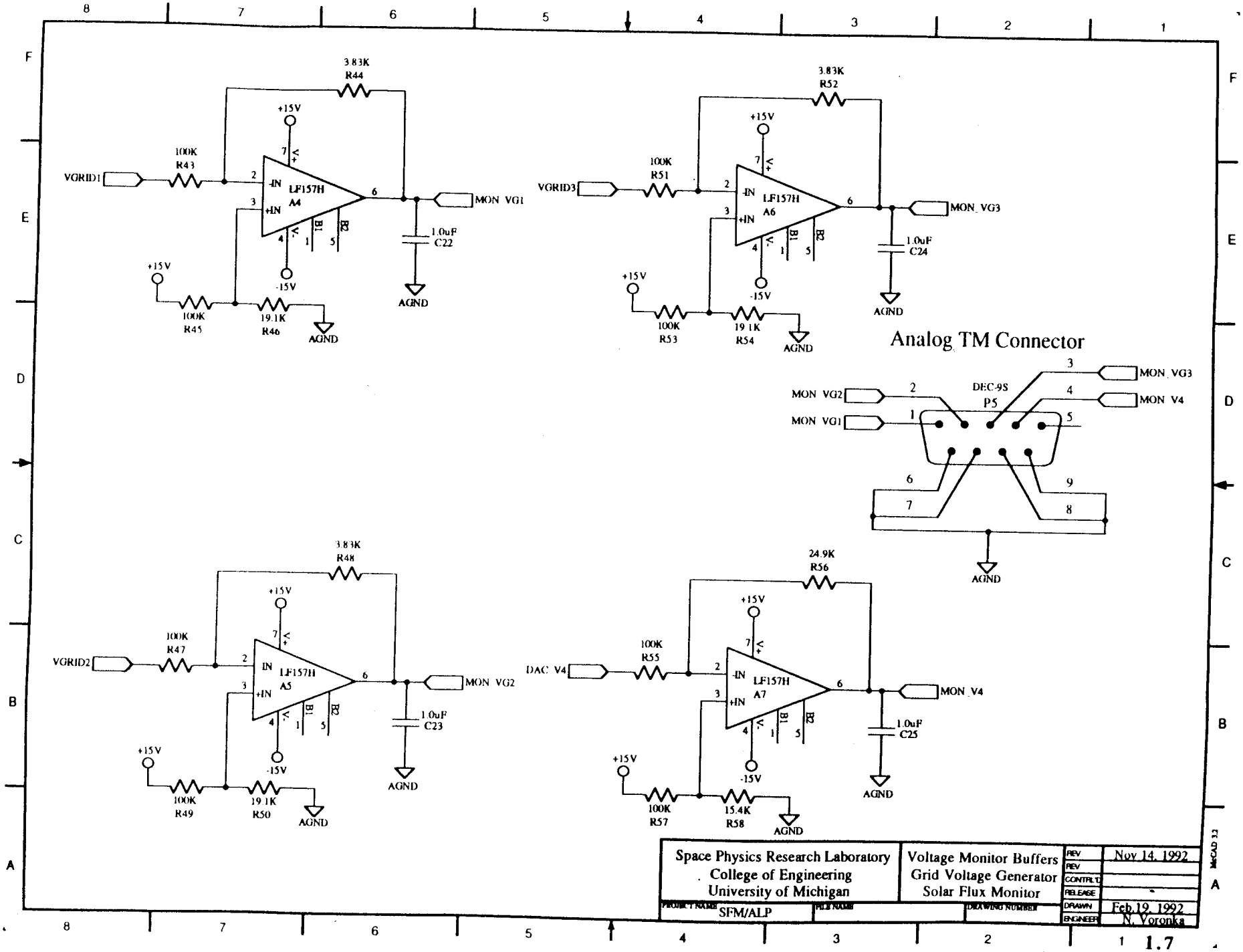


To write to:	Data must be of the form
Programmable Timer	00XXXXAADDDDDDDD
Quadruple DAC	01AADDDDDDDDDDDD
Octal Latch	10XXXXXXDDDDDDDD

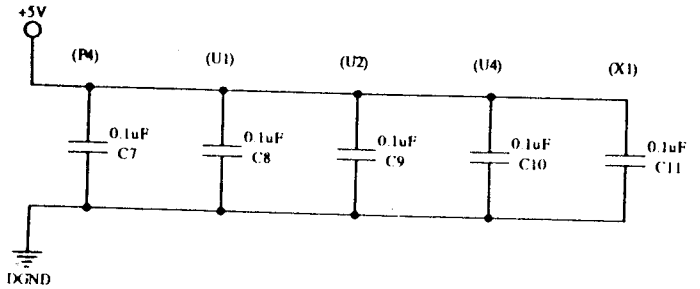
Space Physics Research Laboratory College of Engineering University of Michigan	Programmable Clock Grid Voltage Generator Solar Flux Monitor	REV REV CONTRL'D RELEASE DRAWN Oct. 12, 1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

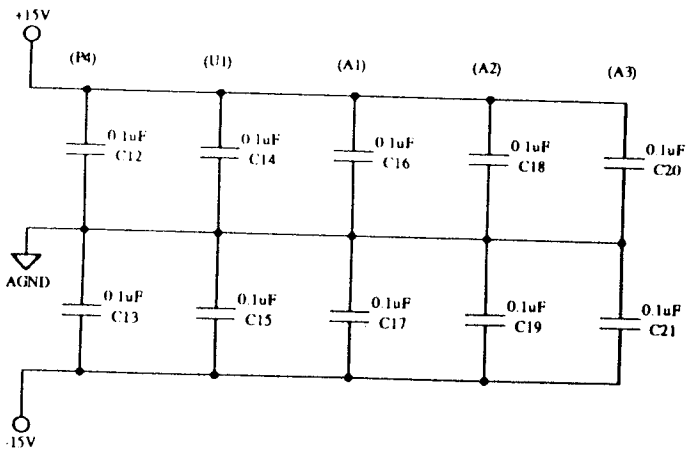
McCAD 3.2

1.6


3.83K
R44
+15V
100K
R43
VGRID1
LF157H
A4
MON VG1
1.0uF
C22
AGND
-15V
100K
R45
19.1K
R46
AGND
3.83K
R52
+15V
100K
R51
VGRID3
LF157H
A6
MON VG3
1.0uF
C24
AGND
-15V
100K
R53
19.1K
R54
AGND
Analog TM Connector
DEC-9S
P5
MON VG2
MON VG1
MON VG3
MON V4
AGND
3.83K
R48
+15V
100K
R47
VGRID2
LF157H
A5
MON VG2
1.0uF
C23
AGND
-15V
100K
R49
19.1K
R50
AGND
24.9K
R56
+15V
100K
R55
DAC V4
LF157H
A7
MON V4
1.0uF
C25
AGND
-15V
100K
R57
15.4K
R58
AGND
Space Physics Research Laboratory
College of Engineering
University of Michigan
Voltage Monitor Buffers
Grid Voltage Generator
Solar Flux Monitor
REV
REV
CONTRL'D
RELEASE
DRAWN
ENGINEER
Nov 14, 1992
Feb 19, 1992
N. Voronka
PROJECT NAME
SFM/ALP
FILE NAME
DRAWING NUMBER
McCAD 3.2

1.7



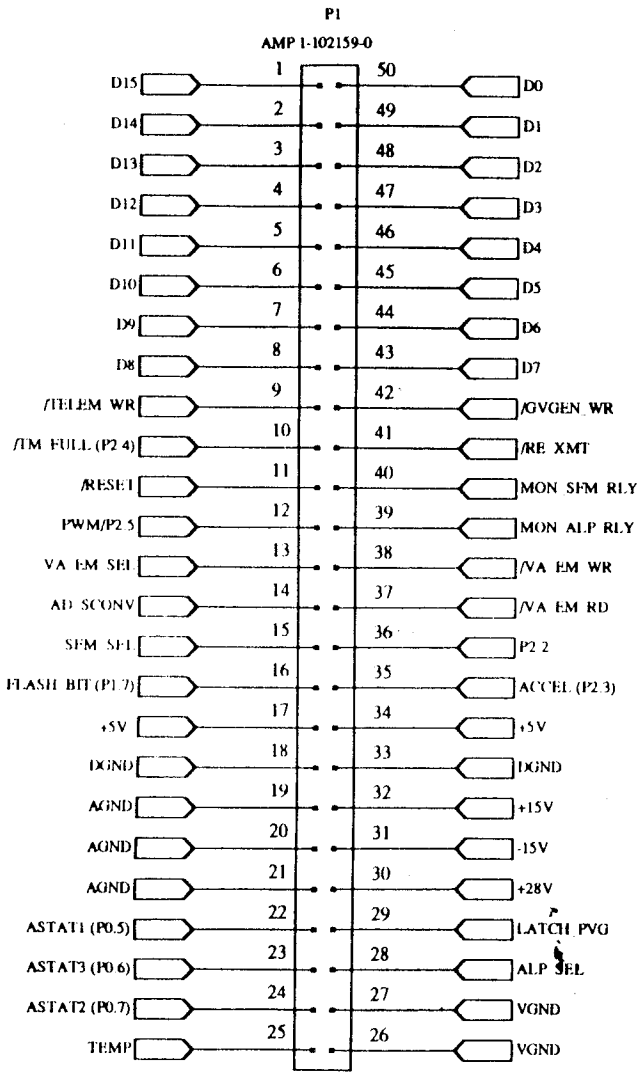


Space Physics Research Laboratory College of Engineering University of Michigan	Power Supply Filters Grid Voltage Generator Solar Flux Monitor	REV REV CONTROL RELEASE DRAWN Feb 20,1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

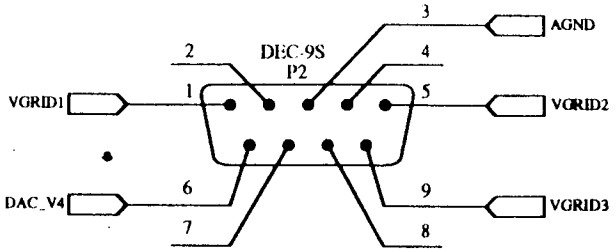
McCAD 12

1.8

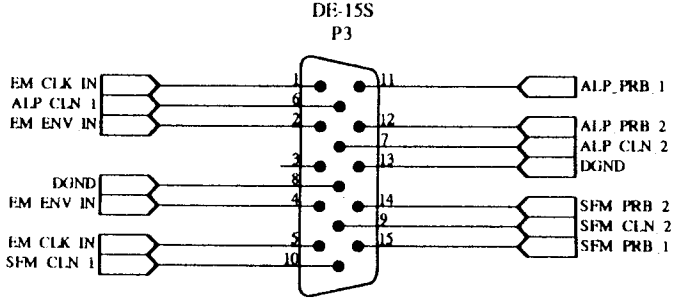
CPU Bus Connector



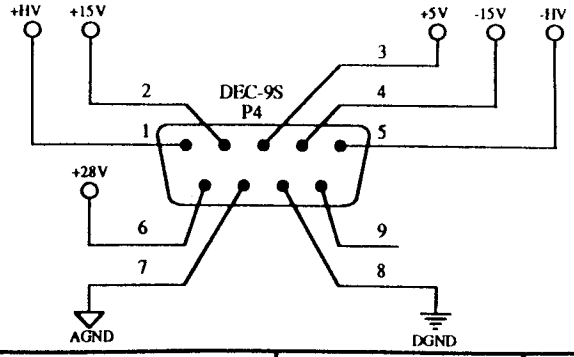
Grid Voltage Connector



Relay Control & Shift Clock Connector



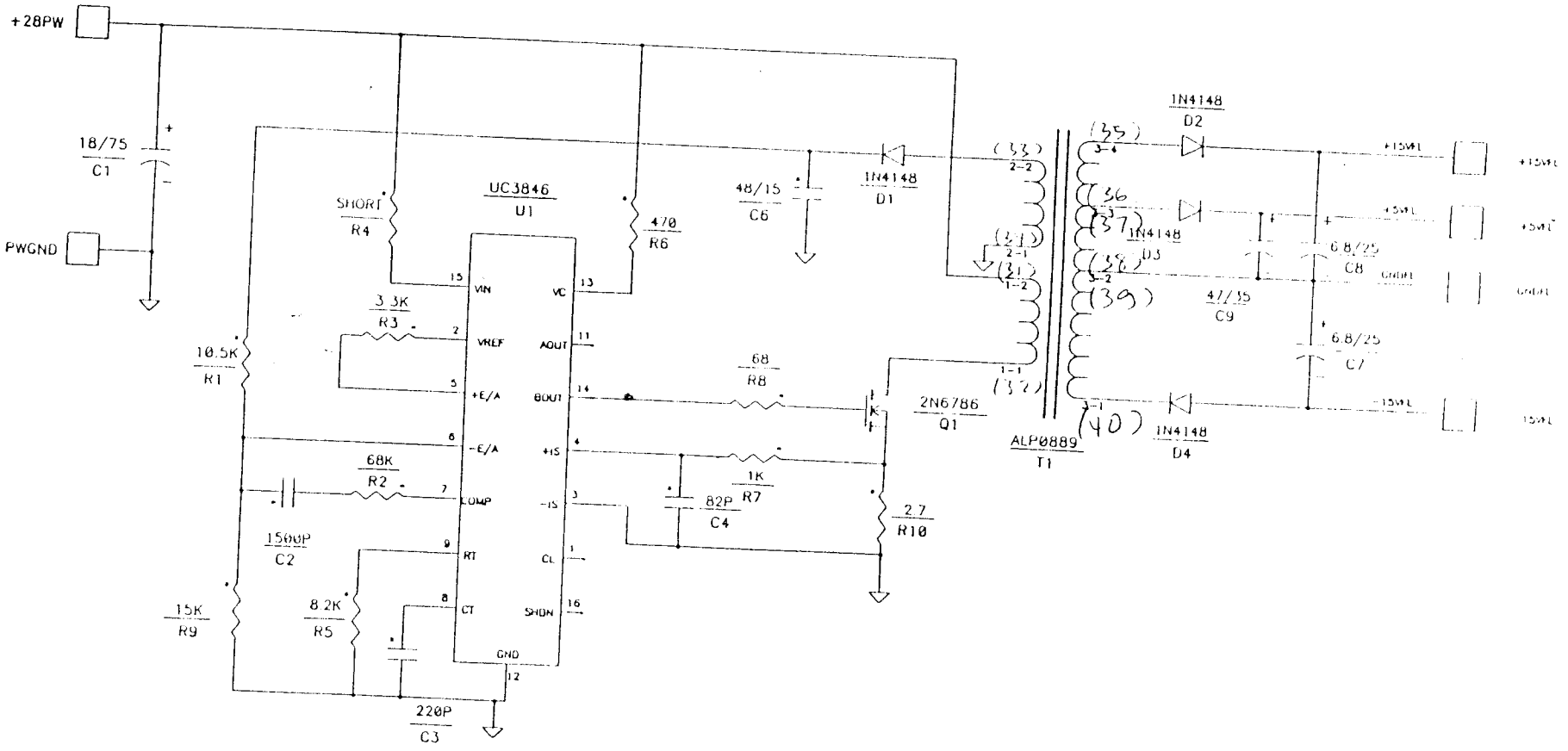
Power Supply Connector



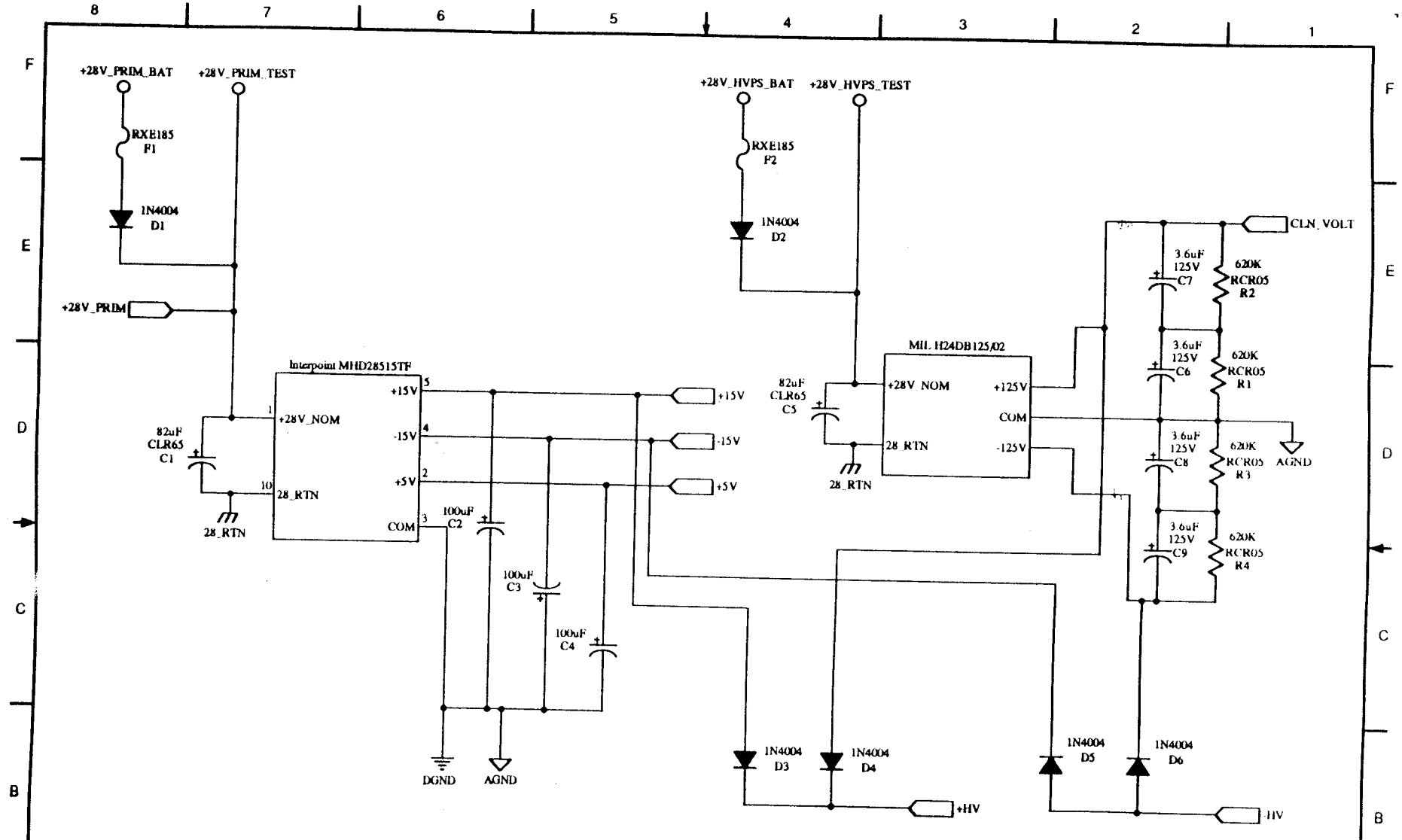
Space Physics Research Laboratory College of Engineering University of Michigan	Connectors Grid Voltage Generator Solar Flux Monitor	REV	Nov 14, 1992
		REV	
		CONTRL'D	
		RELEASE	
PROJECT NAME SFM/ALP	FILE NAME / DRAWING NUMBER	DRAWN	Sept 14,1991
		ENGINEER	N. Voronka

McCAD 12

PWR.SUP DWG.



ENGINEER BLOCK	DRAFTSMAN 8/26/89	
SPACE PHYSICS RESEARCH LABORATORY COLLEGE OF ENGINEERING UNIVERSITY OF MICHIGAN ANN ARBOR, MICHIGAN	+/-15, +5 POWER SUPPLY CURRENT-MODE TECHNOLOGY CRAF-NGIMS/ALP	
	B-F8574	DATE

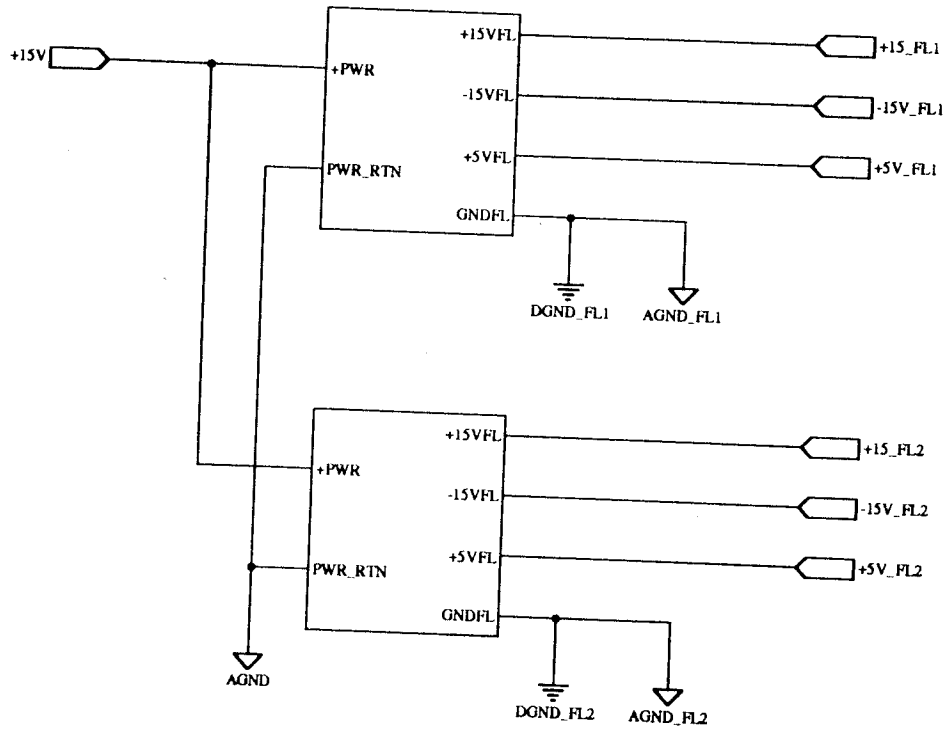


NOTES:

1. Devices F1 and F2 are PolySwitch current limiting devices. They were required by NASA/WFF for the 27.133 rocket flight experiment. They will sustain a maximum of 1.85 amperes without tripping.

Space Physics Research Laboratory College of Engineering University of Michigan	Primary&HV Converters Solar Flux Monitor /Adv. Langmuir Probe	REV	Nov 14, 1991
		REV	
		CONTROL	
		RELEASE	
PROJECT NAME SFM/ALP	PLOT NAME	DRAWN	Mar 5, 1992
	DRAWING NUMBER	ENGINEER	N. Voronka

McCAD 3.3

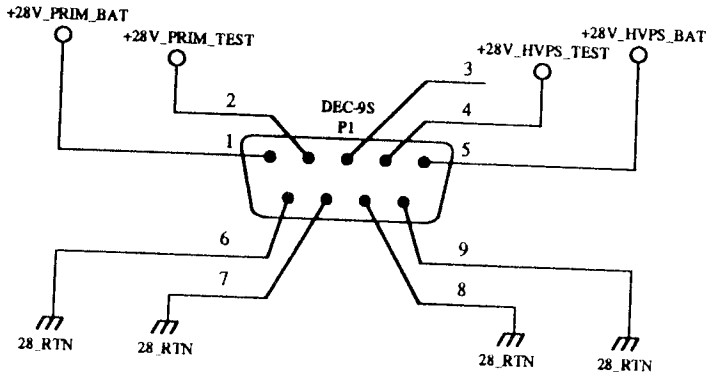


Space Physics Research Laboratory College of Engineering University of Michigan	Floating PS Converters Solar Flux Monitor /Adv. Langmuir Probe	REV	
		REV	
		CONTROL	
		RELEASE	
PROJECT NAME: SFM/ALP	FILE NAME	DRAWN	Mar 7, 1992
	DRAWING NUMBER	ENGINEER	N. Voronka

McCAD 3.2

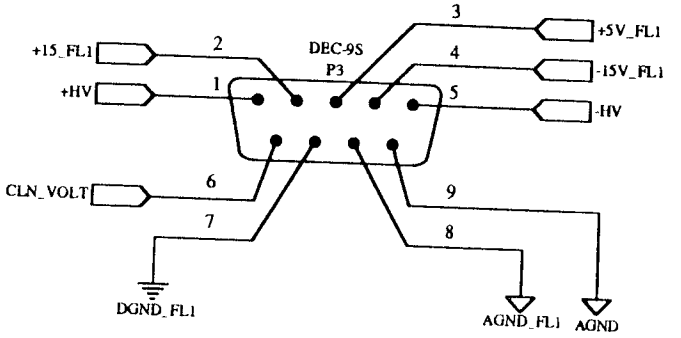
3.2

Primary Power Connector (to SC)



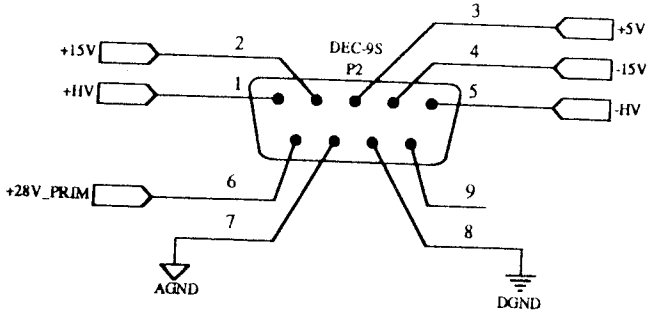
Power Supply Connector

ALP VA Gen./Electrometer



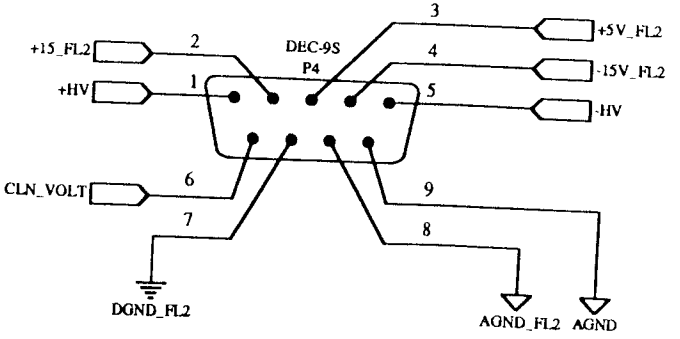
Power Supply Connector

Grid Voltage Generator



Power Supply Connector

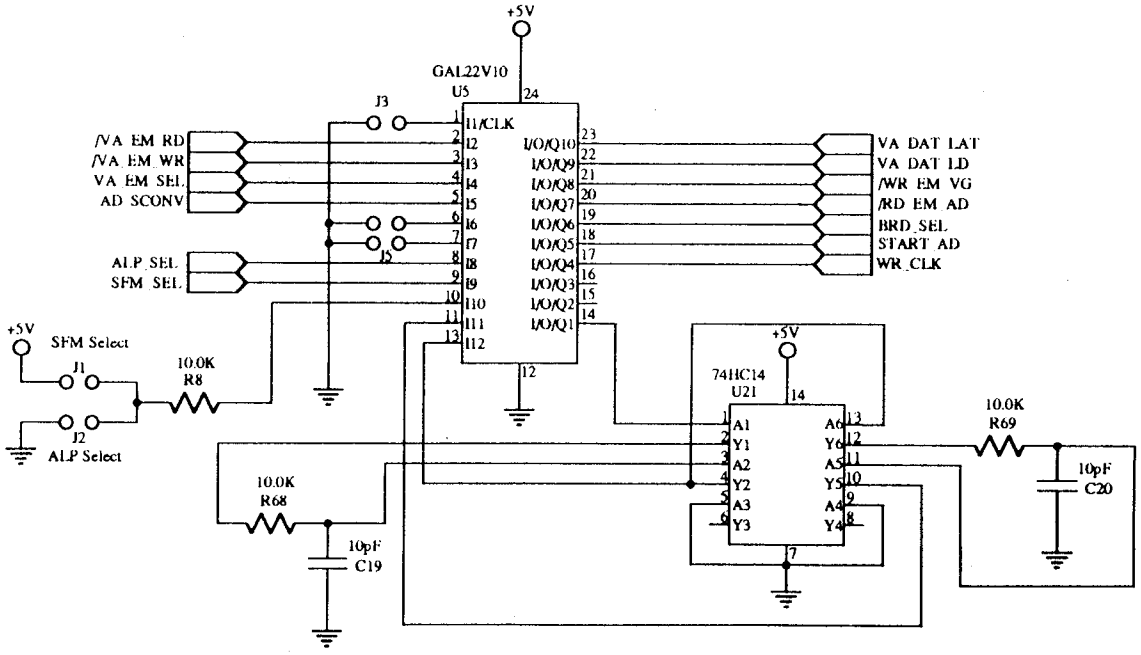
SFM VA Gen./Electrometer



Space Physics Research Laboratory College of Engineering University of Michigan	Powe Supply Connectors Solar Flux Monitor /Adv. Langmuir Probe	REV REV CONTROL RELEASE DRAWN Jan 9, 1992 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

McCAD 3.2

SFM/ALP DWG.

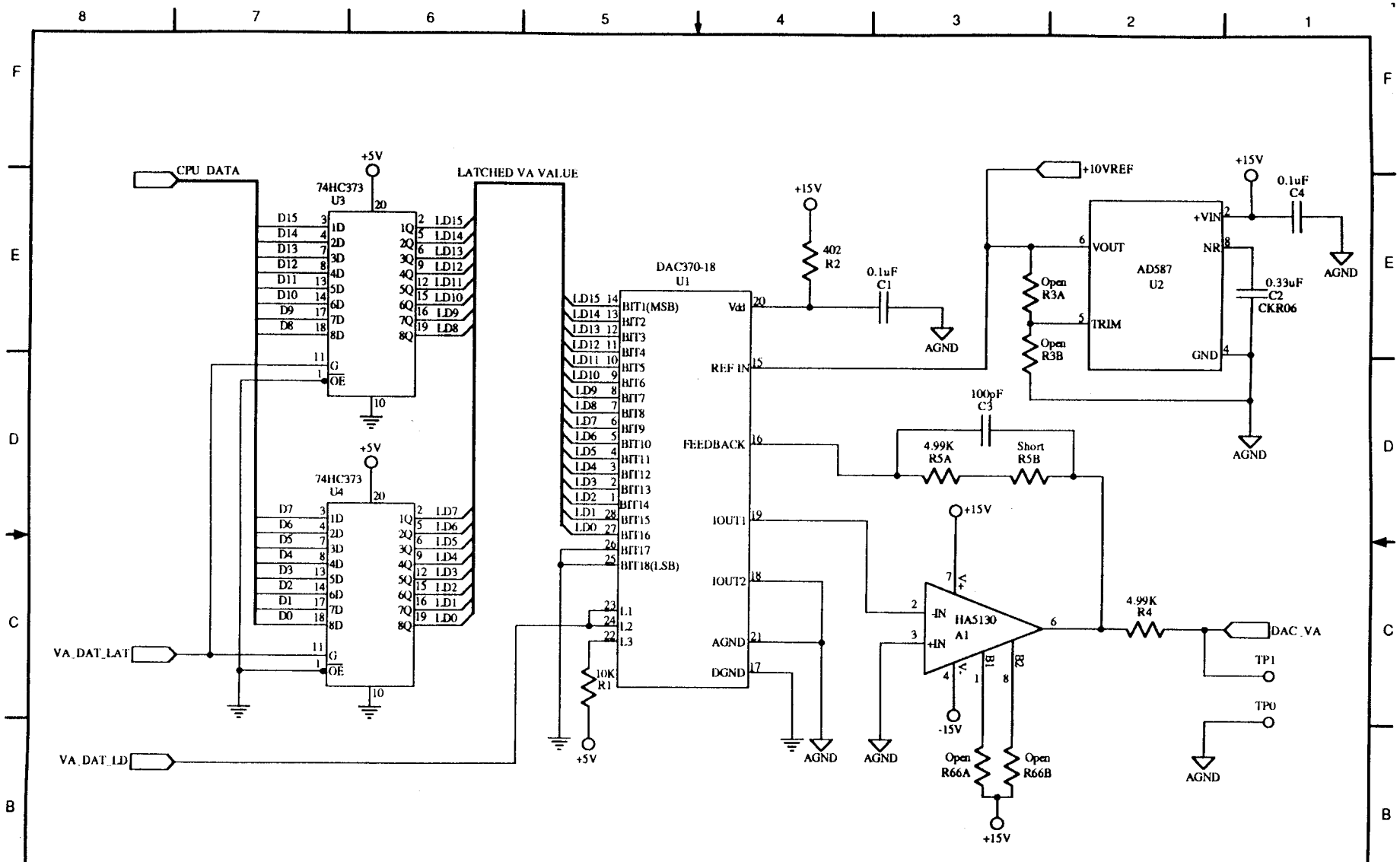


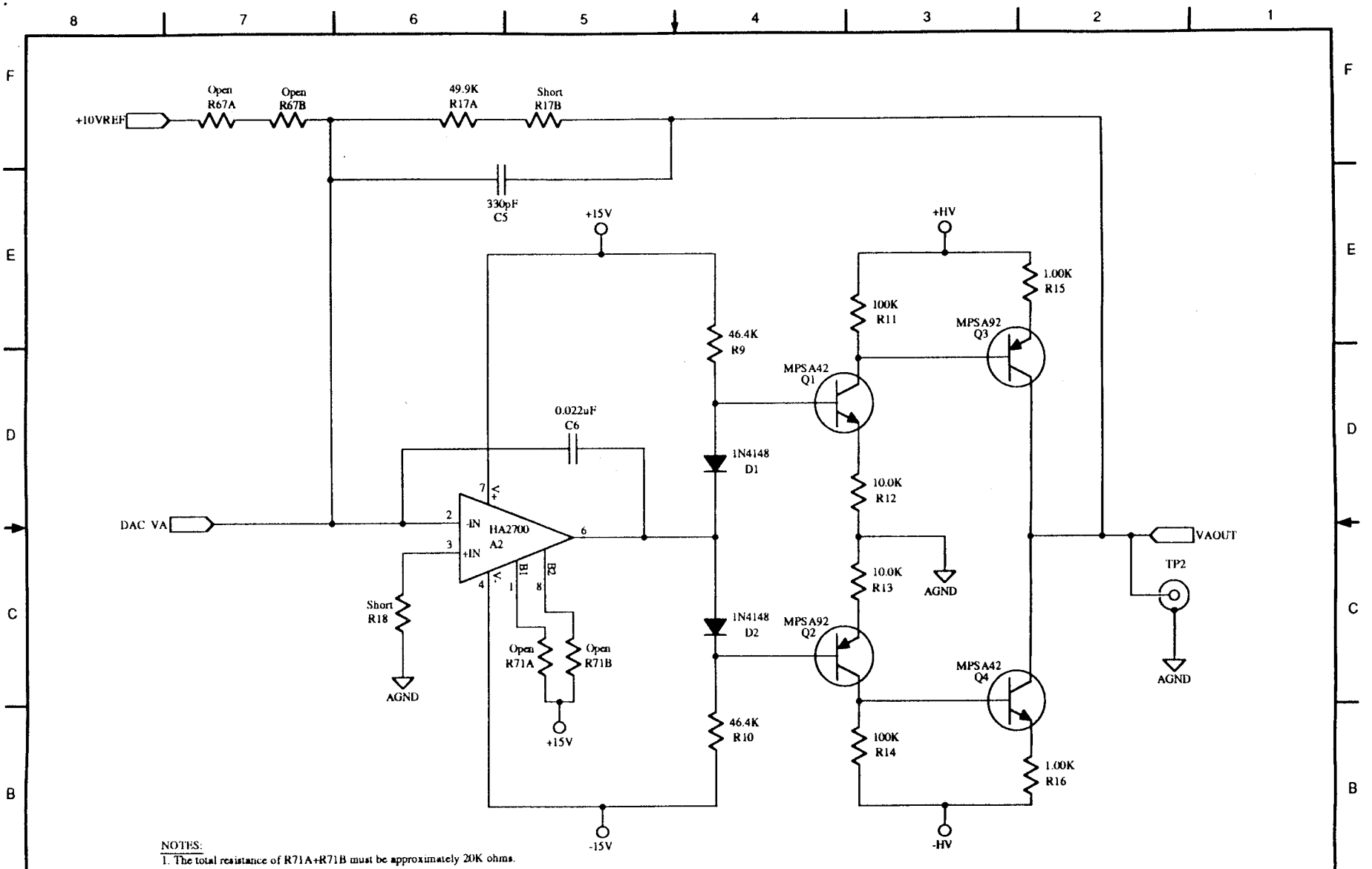
NOTES:
1. Unless otherwise specified, all resistors are type RNC55 and all capacitors are type CKR05.

Space Physics Research Laboratory College of Engineering University of Michigan	Control Logic VA Gen./Electrometer SFM/ALP	REV	Nov 14, 1991
		REV	
		CONTRL'D	
		RELEASE	
PROJECT NAME: SFM/ALP	PILE NAME / DRAWING NUMBER	DRAWN	Sept 19, 1991
		ENGINEER	N. Voronka

McCAD 3.2

2.01


CPU DATA
74HC373
U3
+5V
LATCHED VA VALUE
74HC373
U4
VA_DAT_LAT
VA_DAT_LD
DAC370-18
U1
BIT1(MSB)
BIT18(LSB)
Vdd
REF IN
FEEDBACK
IOUT1
IOUT2
AGND
DGND
402
R2
0.1uF
C1
10K
R1
+10VREF
AD587
U2
VOUT
TRIM
+VIN
NR
GND
0.1uF
C4
0.33uF
C2
CKR06
Open
R3A
Open
R3B
100pF
C3
4.99K
R5A
Short
R5B
HA5130
A1
-IN
+IN
V+
V-
B1
B2
-15V
Open
R66A
Open
R66B
4.99K
R4
DAC_VA
TP1
TP0
NOTES:
1. The total resistance of R3A+R3B must be approximately 10K ohms.
2. The total resistance of R66A+R66B must be approximately 20K ohms.
Space Physics Research Laboratory
College of Engineering
University of Michigan
VA Generator DAC
VA Gen./Electrometer
SFM/ALP
REV
Nov 14, 1991
CONTRL'D
RELEASE
DRAWN
Sept 19, 1991
ENGINEER
Block/Voronka
PROJECT NAME
SFM/ALP
FILE NAME
DRAWING NUMBER
McCAD 3.2
2.02

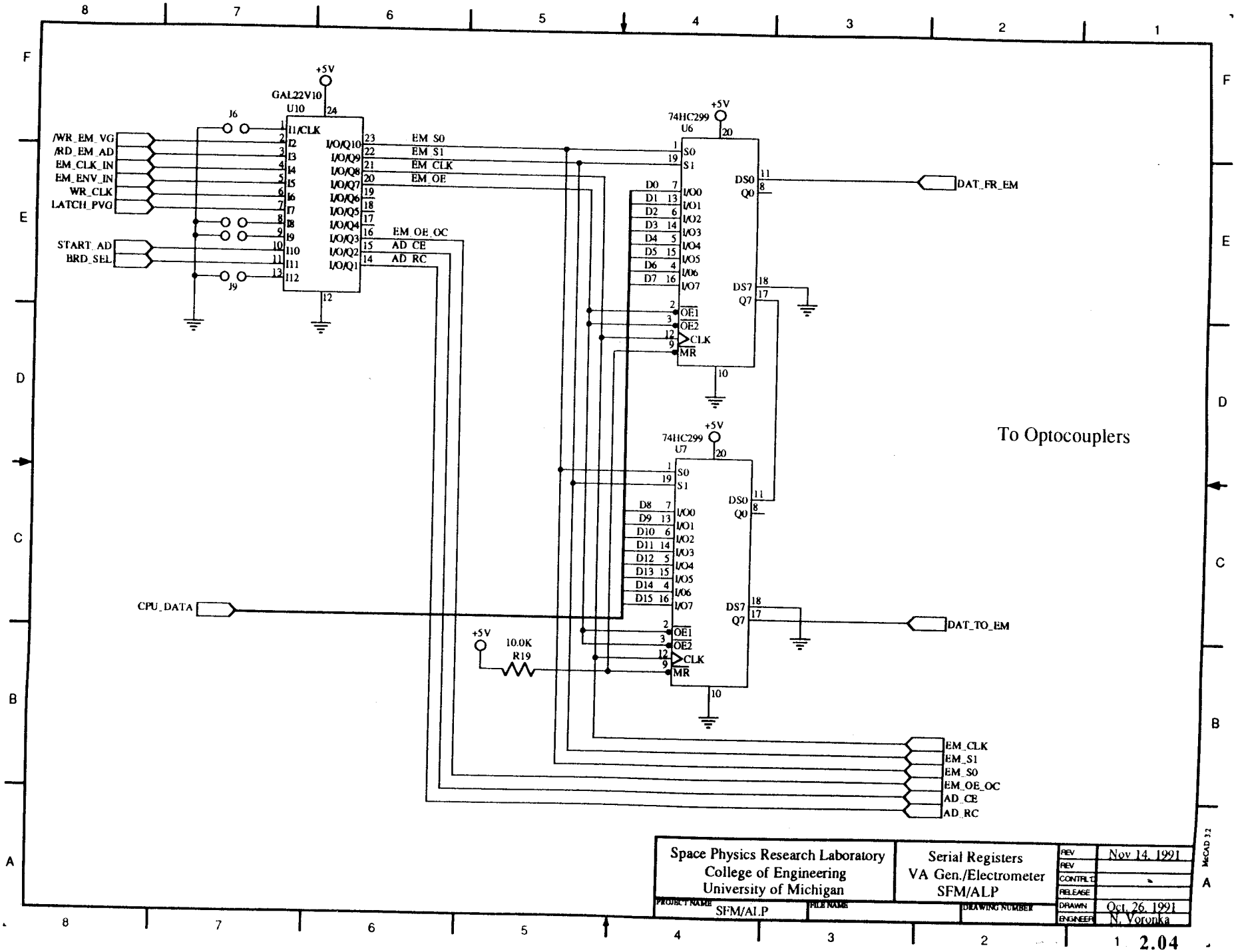


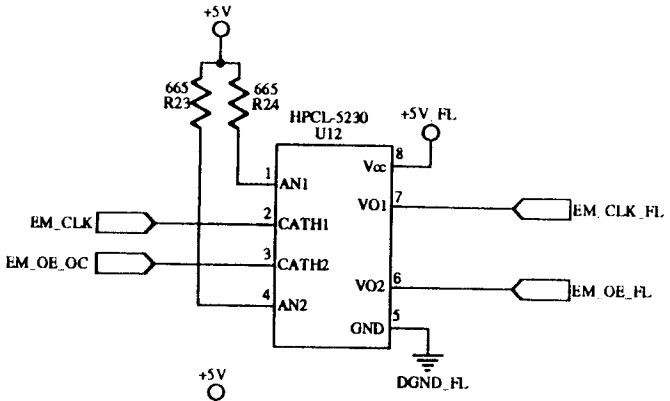
NOTES:
1. The total resistance of R71A+R71B must be approximately 20K ohms.

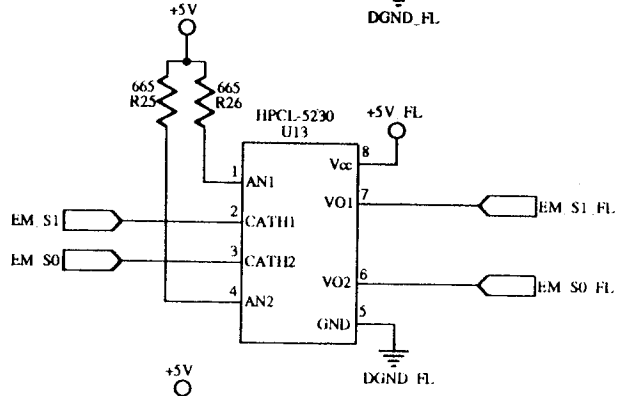
Space Physics Research Laboratory College of Engineering University of Michigan	VA Bipolar Amplifier VA Gen./Electrometer SFM/ALP	REV	Nov 14, 1991
		REV	
		CONTROL	
		RELEASE	
PROJECT NAME: SFM/ALP	FILE NAME	DRAWN	Sept 17, 1991
	DRAWING NUMBER	ENGINEER	Block/Voronka

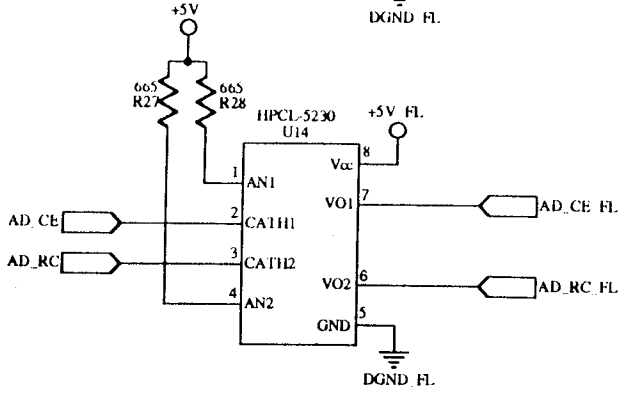
McCAD 3.2

2.03

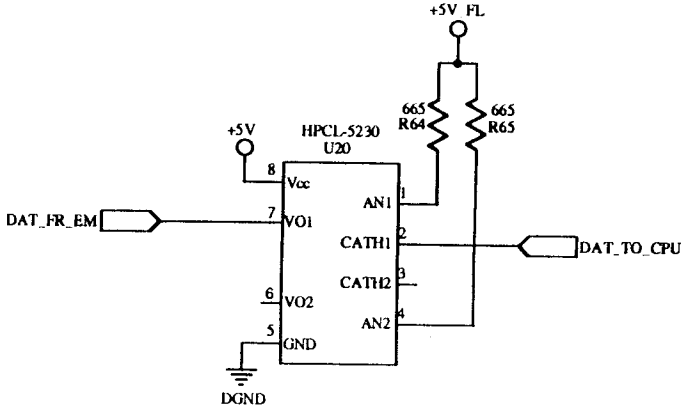

GAL22V10
U10
+5V
J6
J9
/WR_EM_VG
/RD_EM_AD
EM_CLK_IN
EM_ENV_IN
WR_CLK
LATCH_PVG
START_AD
BRD_SEL
EM_S0
EM_S1
EM_CLK
EM_OE
EM_OE_OC
AD_CE
AD_RC
74HC299
U6
U7
DAT_FR_EM
DAT_TO_EM
CPU_DATA
10.0K
R19
To Optocouplers
EM_CLK
EM_S1
EM_S0
EM_OE_OC
AD_CE
AD_RC
Space Physics Research Laboratory
College of Engineering
University of Michigan
Serial Registers
VA Gen./Electrometer
SFM/ALP
PROJECT NAME
SFM/ALP
FILE NAME
DRAWING NUMBER
REV
Nov 14, 1991
CONTROL
RELEASE
DRAWN
Oct. 26, 1991
ENGINEER
N. Voronka
McCAD 3.2
2.04

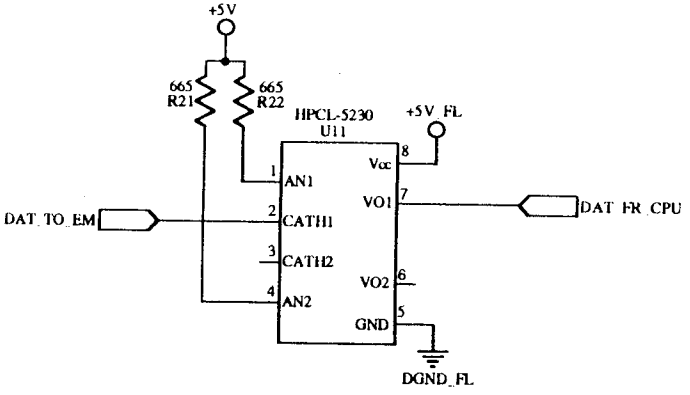






Control Signals



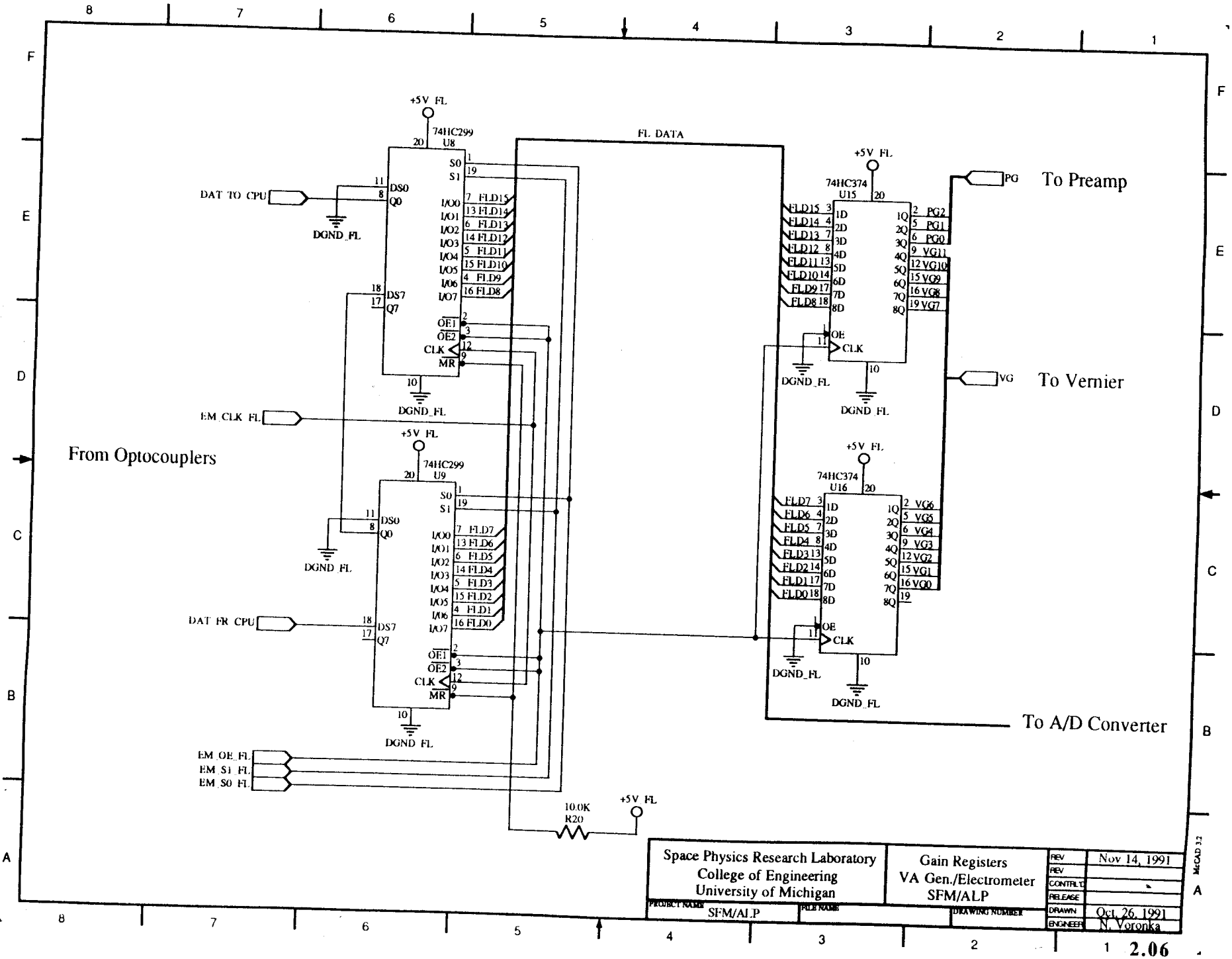


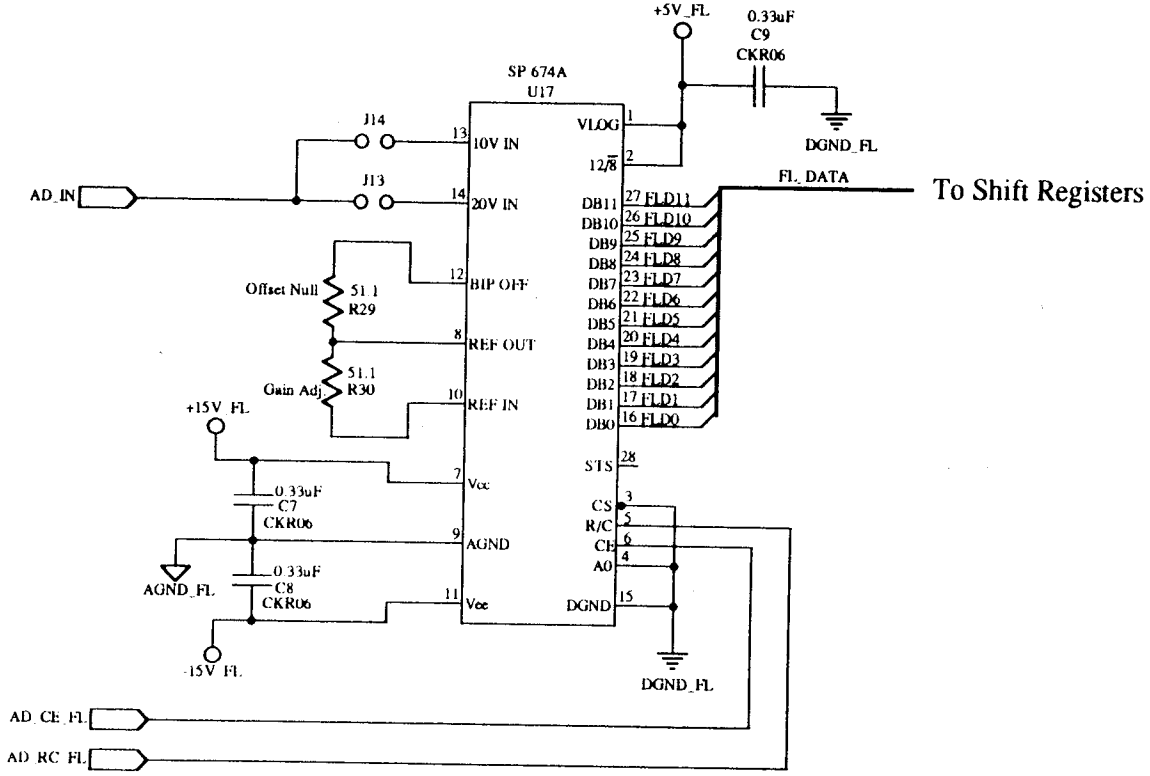
Data Signals

Space Physics Research Laboratory College of Engineering University of Michigan	Optocoupler Interface VA Gen./Electrometer SFM/ALP	REV	
		REV	
		CONTRLT	
		RELEASE	
PROJECT NAME: SFM/ALP	FILE NAME	DRAWN	Oct. 26, 1991
	DRAWING NUMBER	ENGINEER	N. Voronka

McCAD 3.2

2.05


+5V_FL
74HC299
U8
DAT_TO_CPU
DGND_FL
EM_CLK_FL
From Optocouplers
74HC299
U9
DAT_FR_CPU
EM_OE_FL
EM_S1_FL
EM_S0_FL
FL_DATA
10.0K
R20
74HC374
U15
74HC374
U16
PG
To Preamp
VG
To Vernier
To A/D Converter
Space Physics Research Laboratory
College of Engineering
University of Michigan
Gain Registers
VA Gen./Electrometer
SFM/ALP
REV
Nov 14, 1991
CONTRLT
RELEASE
DRAWN
Oct. 26, 1991
ENGINEER
N. Voronka
PROJECT NAME
SFM/ALP
FILE NAME
DRAWING NUMBER
2.06



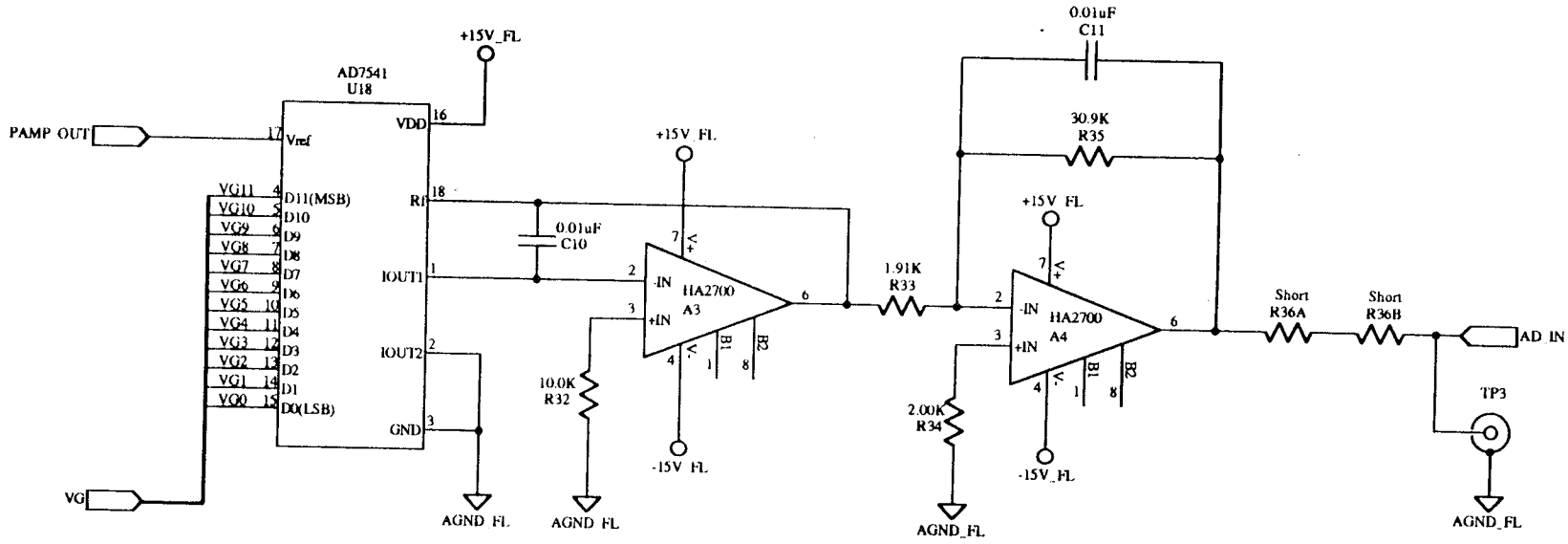
A/D Converter

NOTES:
1. The values of resistors R29 and R30 must not exceed 100 ohms.

Space Physics Research Laboratory College of Engineering University of Michigan	Electrometer A/D VA Gen./Electrometer SFM/ALP	REV Nov 14, 1991 REV CONTROL RELEASE DRAWN Oct. 11, 1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

McCAD 3.2

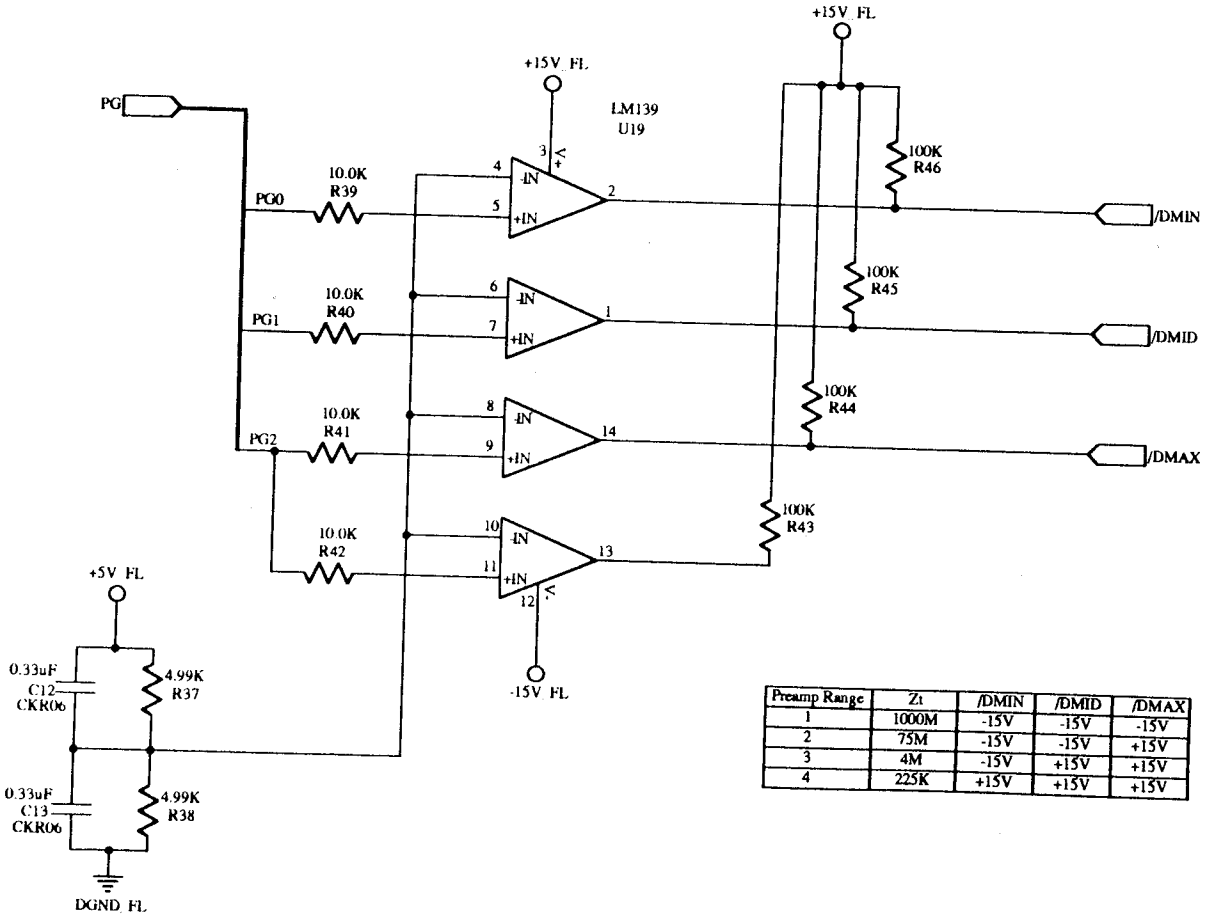
2.07



Space Physics Research Laboratory College of Engineering University of Michigan	Electrometer Vernier VA Gen./Electrometer SFM/ALP	REV	Nov 14, 1991
		REV	
		CONTROL	
		RELEASE	
PROJECT NAME: SFM/ALP	FILE NAME	DRAWN	Oct. 11, 1991
	DRAWING NUMBER	ENGINEER	Block/Voronka

McCAD 32

2.08



Preamp Range	Zt	/DMIN	/DMID	/DMAX
1	1000M	-15V	-15V	-15V
2	75M	-15V	-15V	+15V
3	4M	-15V	+15V	+15V
4	225K	+15V	+15V	+15V

Space Physics Research Laboratory
College of Engineering
University of Michigan

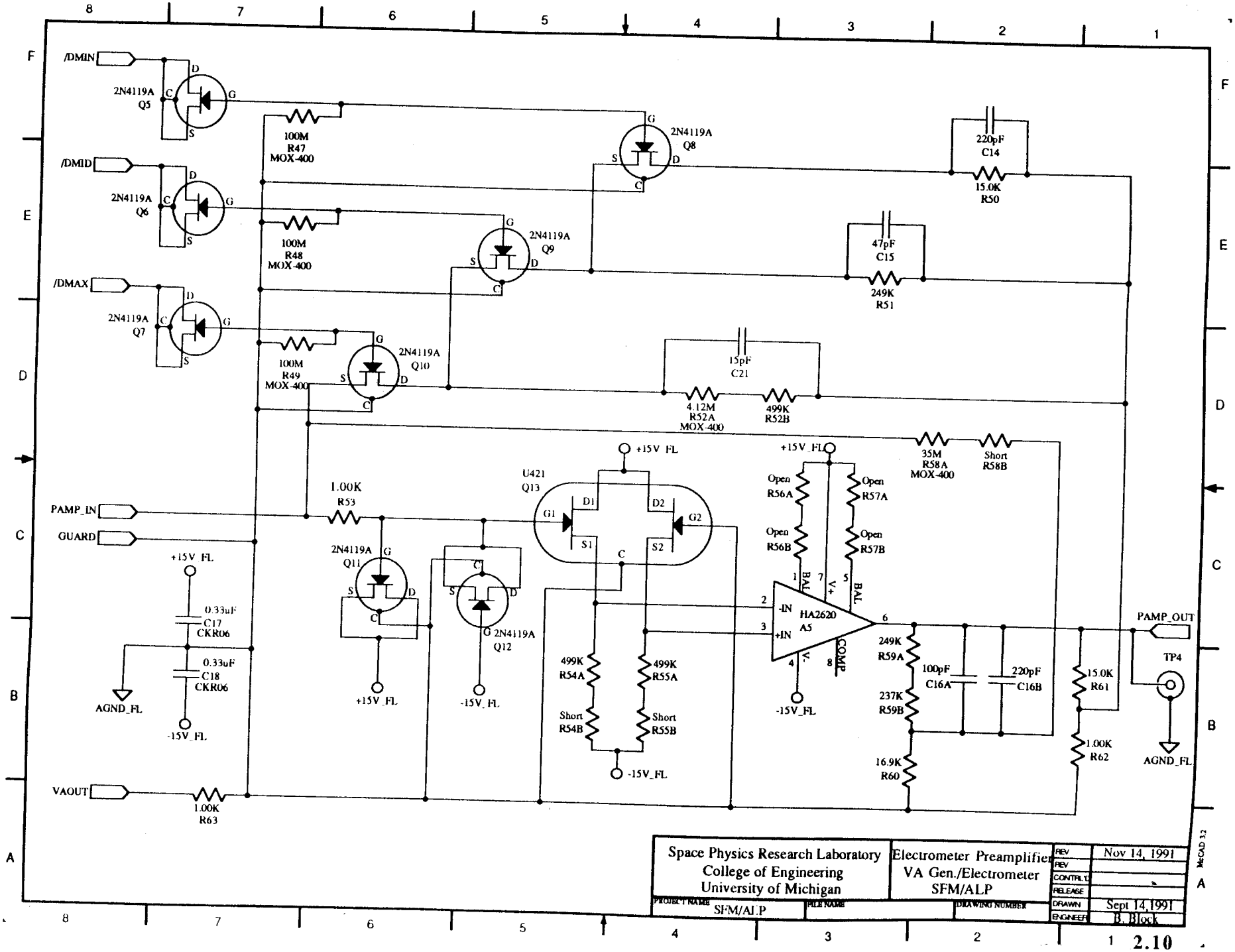
Gain Level Shifters
VA Gen./Electrometer
SFM/ALP

PROJECT NAME: SFM/ALP
FILE NAME:
DRAWING NUMBER:

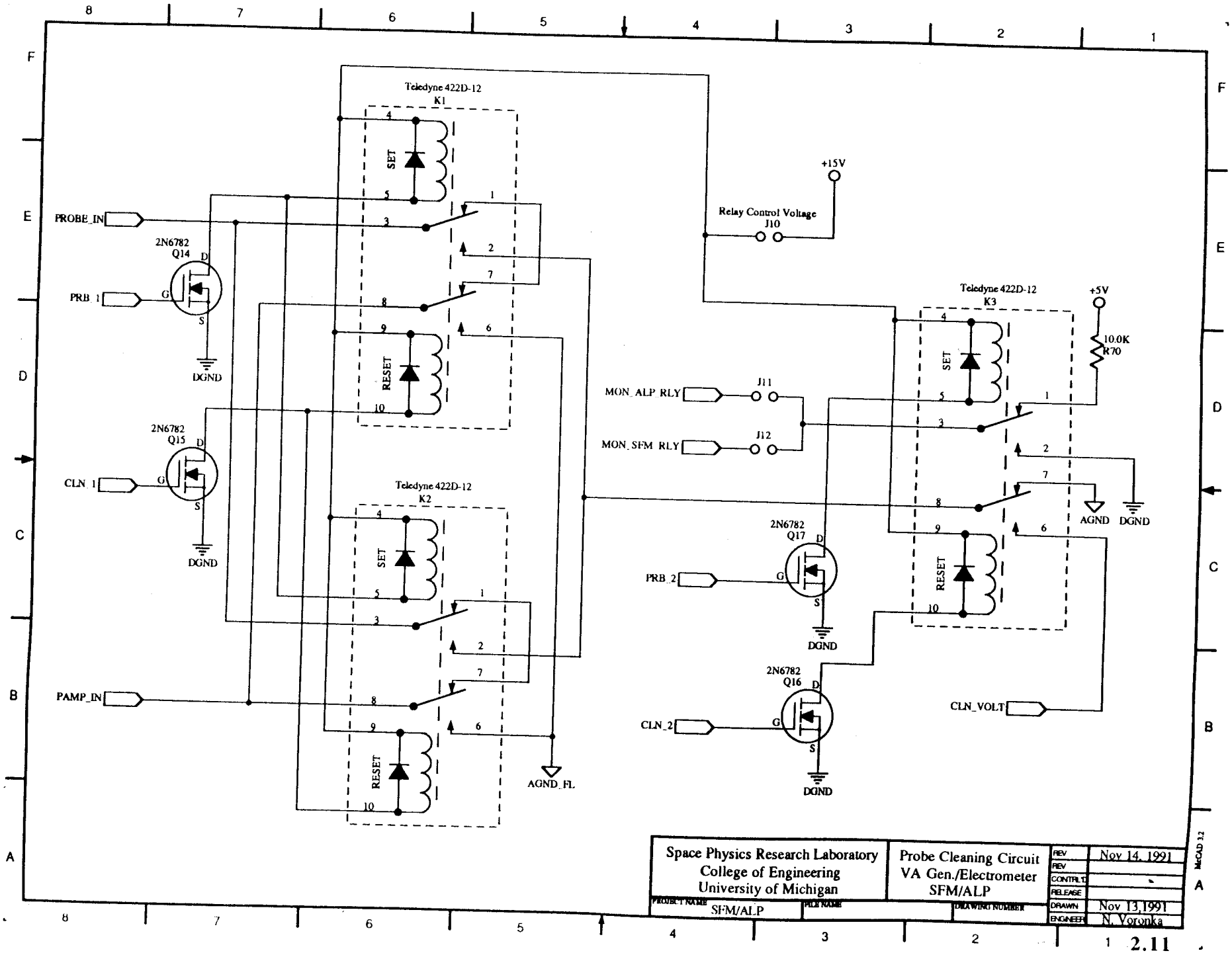
REV
REV
CONTROL
RELEASE
DRAWN: Oct. 11, 1991
ENGINEER: Block/Voronka

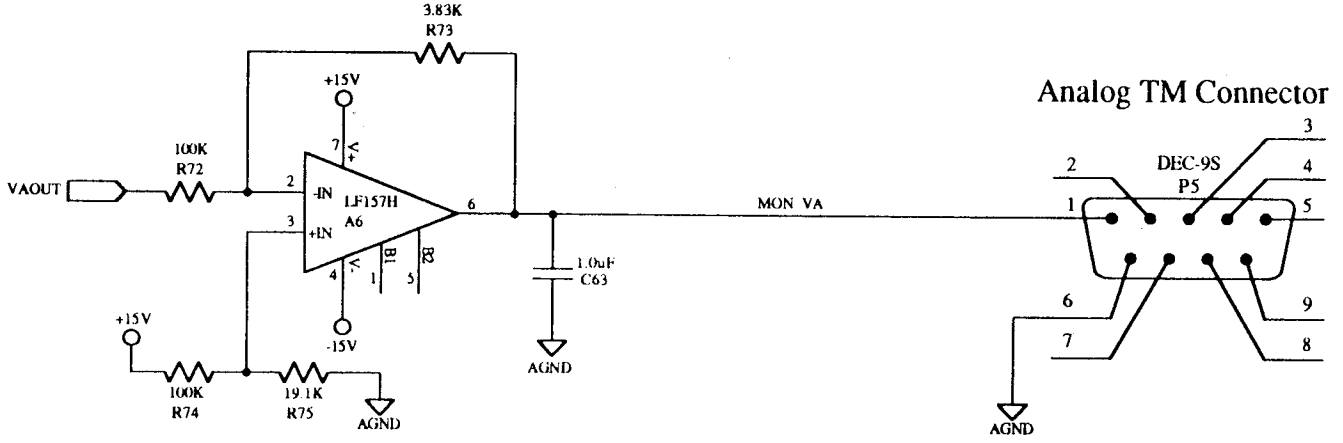
McCAD 3.2

2.09


/DMIN
/DMID
/DMAX
PAMP_IN
GUARD
VAOUT
2N4119A Q5
2N4119A Q6
2N4119A Q7
2N4119A Q8
2N4119A Q9
2N4119A Q10
2N4119A Q11
2N4119A Q12
100M R47 MOX-400
100M R48 MOX-400
100M R49 MOX-400
220pF C14
15.0K R50
47pF C15
249K R51
15pF C21
4.12M R52A MOX-400
499K R52B
35M R58A MOX-400
Short R58B
1.00K R53
U421 Q13
+15V_FL
-15V_FL
AGND_FL
0.33uF C17 CKR06
0.33uF C18 CKR06
499K R54A
Short R54B
499K R55A
Short R55B
Open R56A
Open R56B
Open R57A
Open R57B
HA2620 A5
-IN
+IN
V+
V-
BAL
COMP
249K R59A
237K R59B
16.9K R60
100pF C16A
220pF C16B
15.0K R61
1.00K R62
1.00K R63
PAMP_OUT
TP4
Space Physics Research Laboratory
College of Engineering
University of Michigan
Electrometer Preamplifier
VA Gen./Electrometer
SFM/ALP
PROJECT NAME SFM/ALP
FILE NAME
DRAWING NUMBER
REV Nov 14, 1991
REV
CONTRLTD
RELEASE
DRAWN Sept 14,1991
ENGINEER B. Block
McCAD 3.2

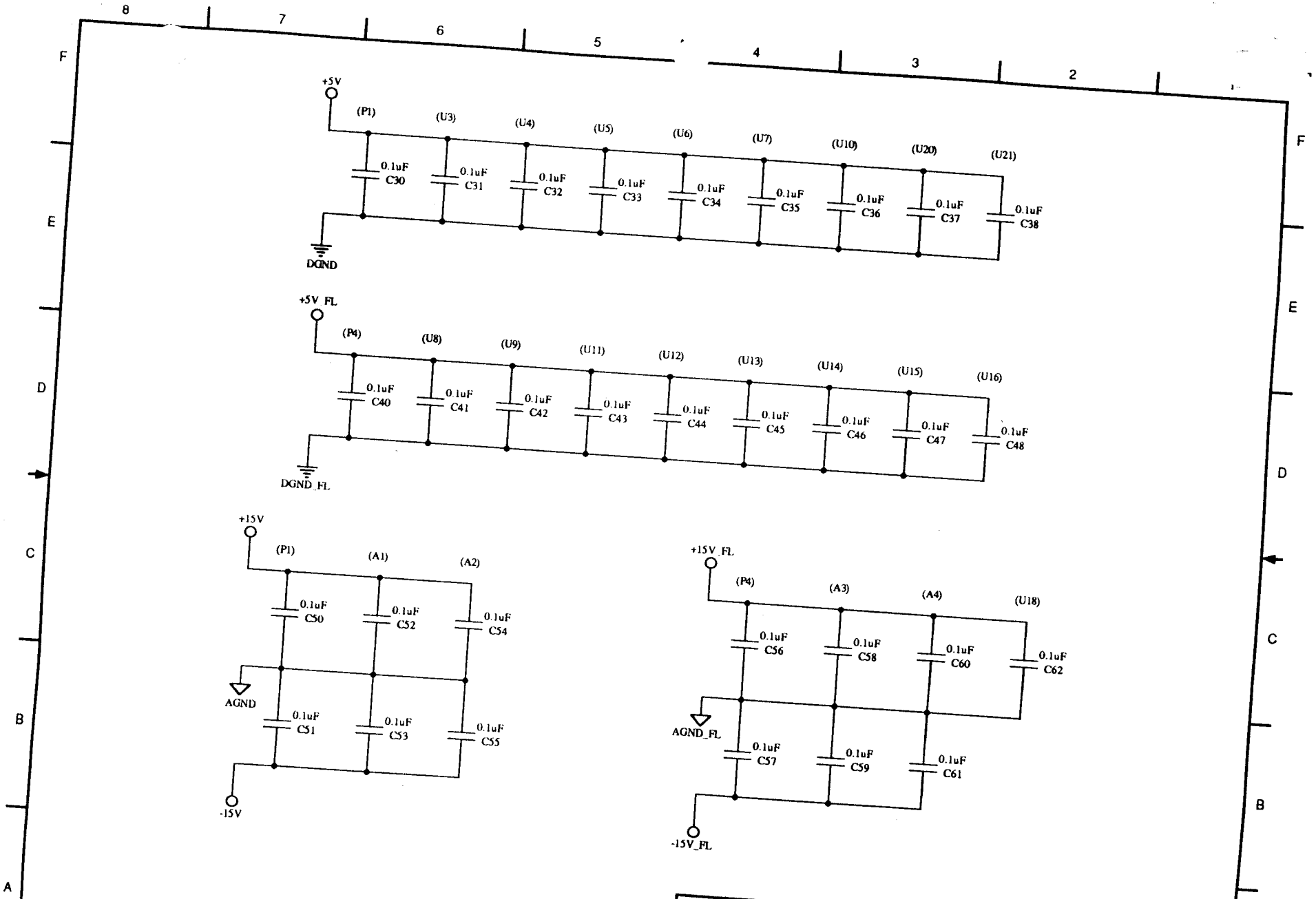
2.10


Teledyne 422D-12
K1
SET
RESET
PROBE_IN
PRB_1
2N6782
Q14
D
G
S
DGND
CLN_1
2N6782
Q15
Teledyne 422D-12
K2
PAMP_IN
AGND_FL
+15V
Relay Control Voltage
J10
MON_ALP_RLY
J11
MON_SFM_RLY
J12
Teledyne 422D-12
K3
+5V
10.0K
R70
AGND
PRB_2
2N6782
Q17
CLN_2
2N6782
Q16
CLN_VOLT
Space Physics Research Laboratory
College of Engineering
University of Michigan
Probe Cleaning Circuit
VA Gen./Electrometer
SFM/ALP
REV
REV
CONTRLT
RELEASE
DRAWN
ENGINEER
Nov 14, 1991
Nov 13,1991
N. Voronka
PROJECT NAME
SFM/ALP
FILE NAME
DRAWING NUMBER
McCAD 3.2



Space Physics Research Laboratory College of Engineering University of Michigan	Voltage Monitor Buffers VA Gen./Electrometer Solar Flux Monitor	REV	Nov 14, 1991
		REV	
		CONTRL'D	
		RELEASE	
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER	DRAWN Feb.19, 1992
			ENGINEER N. Voronka

McCAD 3.2



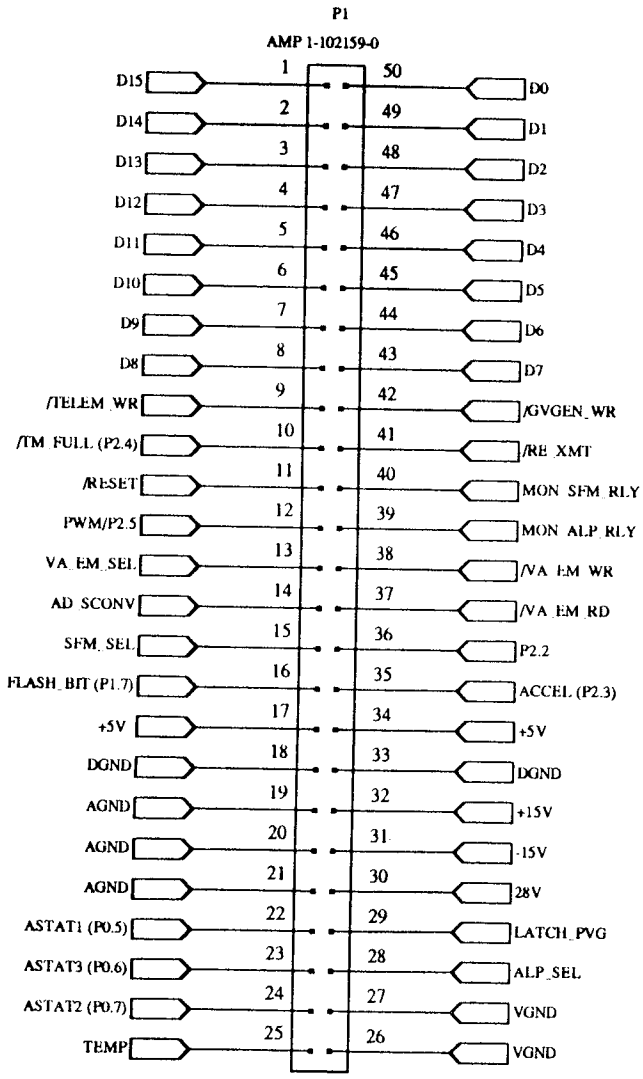
Space Physics Research Laboratory College of Engineering University of Michigan	Power Supply Filters VA Gen./Electrometer Solar Flux Monitor	REV REV CONTRLT RELEASE DRAWN Feb 20,1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

McCAD 32

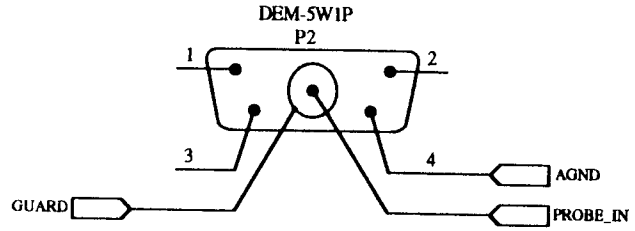
A

2.13

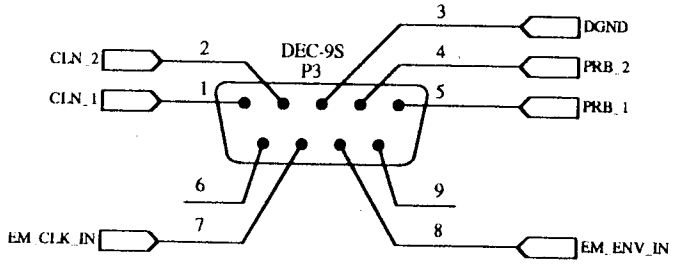
System Connector



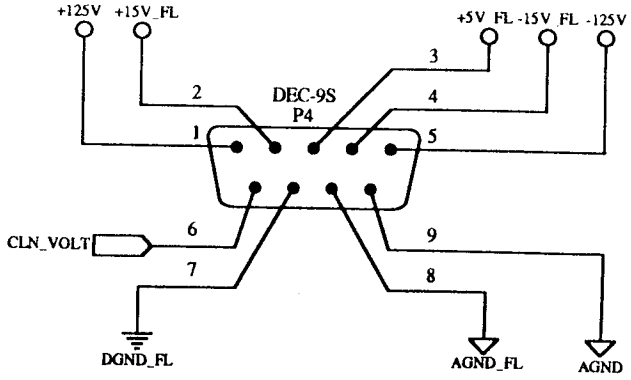
Sensor Connector



Relay Control & Shift Clock Connector



Power Supply Connector

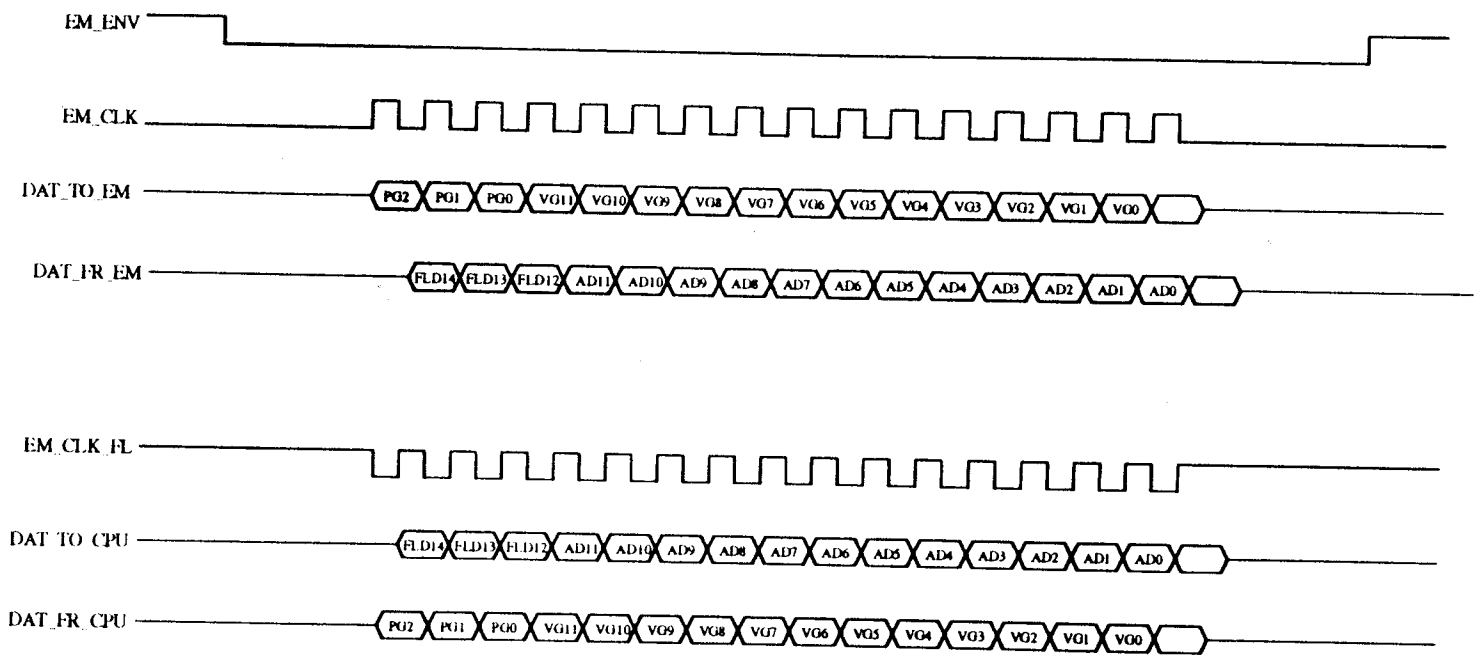


NOTES:
1. A jumper must be installed between pin 3 of P2 (GUARD) and pin 8 of P4 (AGND_FL).

Space Physics Research Laboratory College of Engineering University of Michigan	Connectors VA Gen./Electrometer SFM/ALP	REV REV CONTROL RELEASE DRAWN Sept 14,1991 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

McCAD 3.2

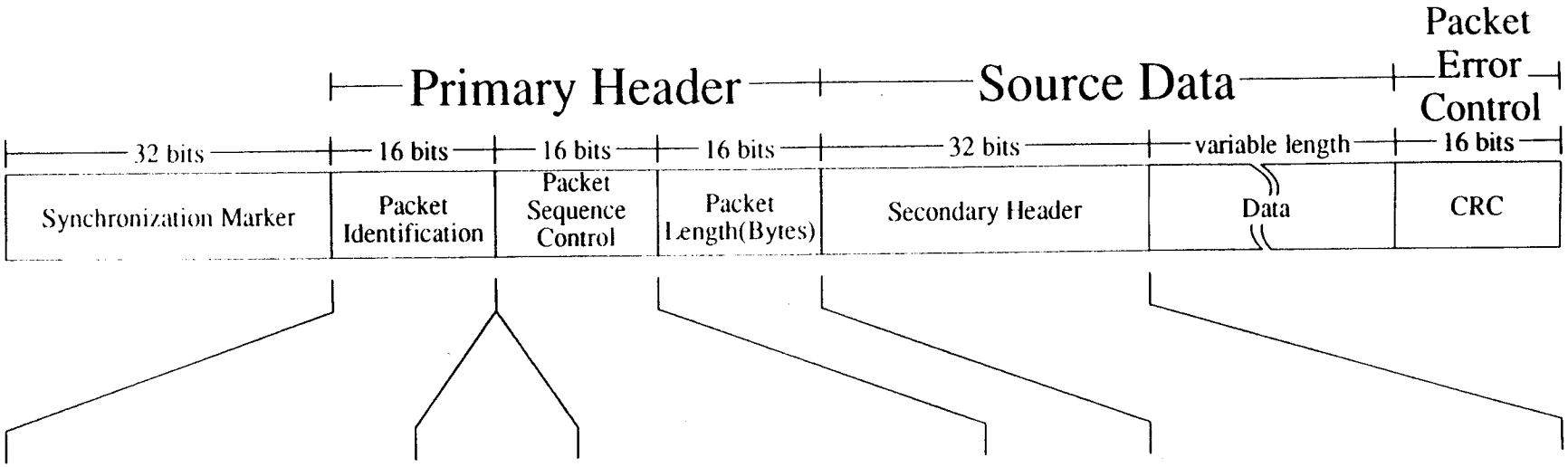
2.14



Space Physics Research Laboratory College of Engineering University of Michigan	Serial Interface Timing Solar Flux Monitor /Adv. Langmuir Probe	REV REV CONTROL RELEASE DRAWN Jan 26,1992 ENGINEER N. Voronka
PROJECT NAME SFM/ALP	FILE NAME	DRAWING NUMBER

2.15

DOCUMENTATION



Packet Identification			
Version #	Packet Type	Secondary Header Flag	Application Process ID
3	1	1	11
000	0	1	00111110000 -- SFM 01000001111 -- ALP 11111111111 -- Idle 10010010101 -- Housekeeping

Packet Sequence Control	
Segmentation Flags	Source Sequence Count
2	14
00-- Continuation Segment 01-- First Segment 10-- Last Segment 11-- Unsegmented Packet	

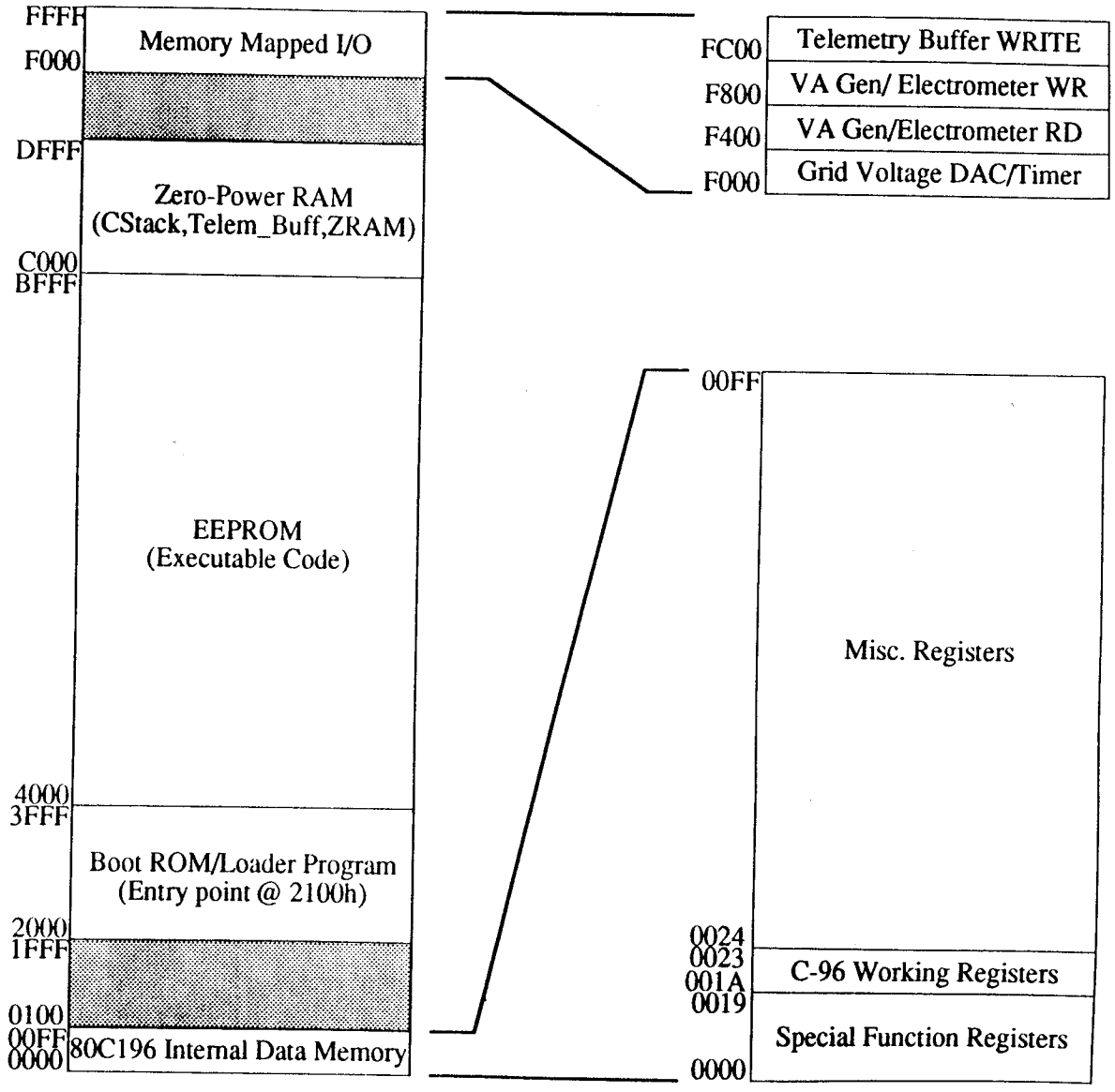
Secondary Header (User Defined)			
Version	Header Length	Mode #	Time Stamp
2	6	8	~~16~~ 32
00	~~000011~~ 000101	x23 -- SFM_STD x2B -- SFM_HR x26 -- SFM_GT x29 -- SFM_GS x27 -- SFM_RPA x2E -- SFM_CLN x85 -- SFM_IRP x87 -- SFM-ERP x8A -- SFM_IH x8C -- SFM_EH x40 -- ALP_5PT x41 -- ALP_STD x42 -- ALP_HR x44 -- ALP_MR x48 -- ALP_CLN	

Notes:

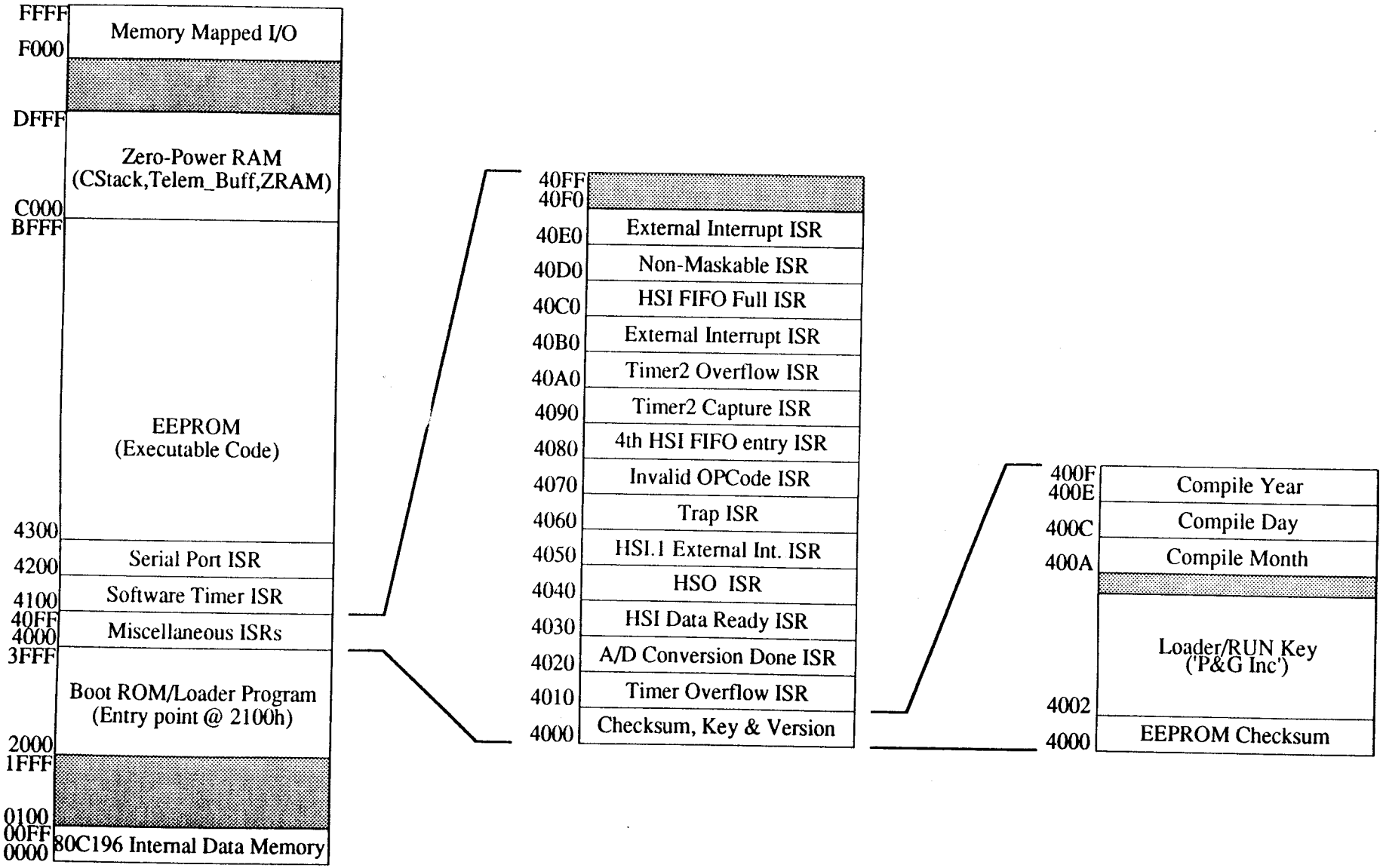
1. The above format conforms to the CCSDS Packet Telemetry format as specified in the CCSDS 102.0-B-2 Blue Book (January 1987).
2. The Frame Synchronization Marker when NOT using Reed-Solomon Encoding is 0x1ACFFC1D.
3. The error control word is a cyclic redundancy code generated using the genenerator polynomial specified in the CCSDS Telemetry 100.0-G-1 Green Book (December 1987.)

Space Physics Research Laboratory College of Engineering University of Michigan	SFM/ALP Packet Telemetry Data Format	RELEASE DRAWN COG ENG N. VORONKA
SFM/ALP	A-E	

MEMORY MAPS



Space Physics Research Laboratory College of Engineering University of Michigan	SFM/ALP Flight Computer Memory Map	RELEASE DRAWN COG ENG N. VORONKA
SFM/ALP	A-E	



Space Physics Research Laboratory College of Engineering University of Michigan	SFM/ALP Flight Computer EEPROM Memory Map	RELEASE DRAWN COG ENG N. VORONKA
SFM/ALP	A-E	

PORT0

P0.7	*ASTAT2*
P0.6	*ASTAT3*
P0.5	*A_PWM*
P0.4	+5V
P0.3	+28V
P0.2	+15V
P0.1	-15V
P0.0	*A_TEMP*

Analog Input Port (10-bit A/D)

PORT1 (000F)

P1.7	FLASH_BIT
P1.6	SFM_SEL
P1.5	ALP_SEL
P1.4	VA_EM_SEL
P1.3	AD_SCONV
P1.2	LATCH_PVG
P1.1	*DSTAT1*
P1.0	*DSTAT0*

Digital Output Port

PORT2 (0010)

P2.7	MON_SFM_RLY
P2.6	MON_ALP_RLY
P2.5	*PWM*
P2.4	/TM_FULL
P2.3	ACCEL
P2.2	*EXT_INT*
P2.1	RXD
P2.0	TXD

Digital Input Port (except P2.5 & P2.0)

SFM_SEL 1 - SFMs VA/EM Board enabled
0 - SFMs VA/EM Board disabled
ALP_SEL 1 - ALPs VA/EM Board enabled
0 - ALPs VA/EM Board disabled
VA_EM_SEL 1 - Electrometer selected
0 - VA Generator selected
AD_SCONV 1 - Start A/D on enabled board(s)
LATCH_PVG 1 - Latch Preamp & Vernier Gains

MON_SFM_RLY 1 - SFM is in cleaning mode
0 - SFM is in measurement mode
MON_ALP_RLY 1 - ALP is in cleaning mode
0 - ALP is in measurement mode
ACCEL 1 - Launch signal enable
0 - Umbilical is connected to rocket

Space Physics Research Laboratory College of Engineering University of Michigan	SFM/ALP Flight Computer CPU Port Memory Map		
SFM/ALP		A–E	DRAWN DG ENG N. VORONKA

MODE DEF.

Modes of Operation

(revised October 7, 1992)

I. Solar Flux Monitor

1) Standard EUV Measurement Mode:

G1 = - 25V
G2 = +25V
G3 = - 15V
Target = sweep from **+20V to -36V** in 1V steps (57 total)
Telemetry: 1 reading of Decade & Vernier settings, 57 A/D values
= 1 word + 57 words = 58 words = 116 bytes

2) High Resolution EUV Measurement Mode:

G1 = - 25V
G2 = +25V
G3 = - 15V
Target = sweep from **+20V to -36V** in 0.25V steps (225 total)
Telemetry: 1 reading of Decade & Vernier settings, 225 A/D values
= 1 word + 225 words = 226 words = 452 bytes

3) Grid 1 Test Mode:

G1 = - 35V, - 30V, - 25V, - 20V, - 15V
G2 = +25V
G3 = - 15V
Target = sweep from **+20V to -36V** in 2V steps (29 total)
Telemetry: 5 * (1 reading of Decade & Vernier settings, 29 A/D values)
= 5 * (1 word + 29 words) = 150 words = 300 bytes

4) Grid 2 Test Mode:

G1 = - 25V
G2 = +15V, +20V, +25V, +30V, +35V
G3 = - 15V
Target = sweep from **+20V to -36V** in 2V steps (29 total)
Telemetry: 5 * (1 reading of Decade & Vernier settings, 29 A/D values)
= 5 * (1 word + 29 words) = 150 words = 300 bytes

5) Grid 3 Test Mode:

G1 = - 25V
G2 = +25V
G3 = - 25V, - 20V, - 15V, - 10V, - 5V
Target = sweep from **+20V to -36V** in 2V steps (29 total)
Telemetry: 5 * (1 reading of Decade & Vernier settings, 29 A/D values)
= 5 * (1 word + 29 words) = 150 words = 300 bytes

6) EUV Ion Retarding Potential Analyzer Mode:

G1 = - 10 V
G2 = sweep from -5V to +5V in 0.2V steps (51 total)
G3 = 0V
Target = 0V
Telemetry: 1 reading of Decade & Vernier settings, 51 A/D values
= 1 word + 51 words = 52 words = 104 bytes

7) EUV Electron Retarding Potential Analyzer Mode:

G1 = +10 V
G2 = sweep from -5V to +5V in 0.2V steps (51 total)
G3 = 0V
Target = 0V
Telemetry: 1 reading of Decade & Vernier settings, 51 A/D values
= 1 word + 51 words = 52 words = 104 bytes

8) EUV Cleaning Mode 1 (5 seconds):

G1 = + 50V
G2 = + 30V
G3 = + 15V
Target = +125V
Telemetry: None.

II. Advanced Langmuir Probe

1) Fast (5-point) ALP Mode:

Probe V_a = values determined by algorithm (5 total)
Telemetry: 1 reading of Decade & Vernier settings, 1 reading of kTe, 5 A/D & D/A values
= 1 word + 1 word + 2 * 5 words = 12 words = 24 bytes

2) Standard ALP Sweep Mode:

Probe V_a = sweep from -5 V to +5 V in 0.1V steps (101 total)
Telemetry: 1 reading of Decade & Vernier settings, 101 A/D values
= 1 (16-bit) word + 101 words = 102 words = 204 bytes

3) High Resolution ALP Sweep Mode:

Probe V_a = sweep from -1.7V to +1.7V in 0.04V steps (86 total)
Telemetry: 1 reading of Decade & Vernier settings, 86 A/D values
= 1 word + 86 words = 87 words = 174 bytes

4) Medium Resolution ALP Sweep Mode:

Probe V_a = sweep from -9kTe to +1kTe in kTe/8 steps (81 total), and a single reading at 2V above 1kTe
Telemetry: 1 reading of Decade & Vernier settings, 1 reading of kTe, 81 A/D values
= 1 word + 1 word + 81 words = 83 words
= 166 bytes

5) ALP Cleaning Mode 1 (5 seconds):

Probe V_a = +125 V
Telemetry: None.

Notes:

1) All data will be transmitted in packet form. The telemetry requiremets above do not include packet headers & terminators (start codes, mode ID, packet length, timestamp, checksum).

2) To avoid unnecessary redundancy, the housekeeping information will be sent in a separate packet.

CALIB.H

```
/*****************************************************************************/
/*                                                                           */
/*   Module:     calib.h                                                     */
/*                                                                           */
/*   Purpose:    The header file contains the calibration information for    */
/*               the grid voltage generator, and both applied voltage        */
/*               generators.                                                 */
/*                                                                           */
/*   Inputs:     None.                                                       */
/*                                                                           */
/*   Outputs:    None.                                                       */
/*                                                                           */
/*   Uses:       Nothing external.                                           */
/*                                                                           */
/*   History:    06_Feb_92   Written.                                        */
/*                                                                           */
/*****************************************************************************/

#define G1_GEN_GAIN       -34.5449083809
#define G1_GEN_OFFSET    2048.000

#define G2_GEN_GAIN       -34.5169132875
#define G2_GEN_OFFSET    2048.000

#define G3_GEN_GAIN       -34.4309878891
#define G3_GEN_OFFSET    2048.000

#define G4_GEN_GAIN       206.031746032
#define G4_GEN_OFFSET    2042.000

#define SFM_VA_GAIN       660.522324629
#define SFM_VA_OFFSET   32564.000

#define ALP_VA_GAIN       661.884316917
#define ALP_VA_OFFSET   32654.000

#ifdef CALIB_C

void c_calib_va (char [], char []) ;
void c_calib_gr (char [], char []) ;
void c_calib_em (char [], char []) ;
void c_calib_vr (char [], char []) ;
void c_va_sweep (char [], char []) ;

#else

extern void c_calib_va (char [], char []) ;
extern void c_calib_gr (char [], char []) ;
extern void c_calib_em (char [], char []) ;
extern void c_calib_vr (char [], char []) ;
extern void c_va_sweep (char [], char []) ;

#endif
```

```
##########################################################################################
#                                                                                        #
#    Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    14/Jul/92  16:27:56 #
#                                                                                        #
#          Target option =  80C196                                                       #
#          Memory model  =  large                                                        #
#          Source file   =  calib.c                                                      #
#          List file     =  calib.lst                                                    #
#          Object file   =  calib.rl5                                                    #
#          ASM file      =  calib.sl5                                                    #
#          Command line  =  calib.c -ml -vl -z -P -e -g -L -F -p52 -A                    #
#                                                                                        #
#                                                  (c) Copyright Archimedes Software Inc. 1990 #
##########################################################################################
```

```
1          /*************************************************************************/
2          /*                                                                       */
3          /*     Name:     calib.c                                                 */
4          /*                                                                       */
5          /*     Params:   None.                                                   */
6          /*                                                                       */
7          /*     Input:    None.                                                   */
8          /*                                                                       */
9          /*     Output:                                                           */
10         /*                                                                       */
11         /*                                                                       */
12         /*     Purpose:                                                          */
13         /*                                                                       */
14         /*                                                                       */
15         /*     Mods:     Jan_12_91   NRV   Authored.                             */
16         /*                                                                       */
17         /*******  ***********************************************************/
18
19         #define CALIB_C TRUE
20
21         #include <stdio.h>
22         #include <stdlib.h>
23         #include <ctype.h>
24         #include <string.h>
25         #include <io80c196.h>
26         #include "types.h"
27         #include "debug.h"
28         #include "io.h"
29         #include "calib.h"
30         #include "com_proc.h"
31
32         void c_calib_va (param1, param2)
33         char param1 [] ;                  /* Parameters of command                */
34         char param2 [] ;                  /* Parameters of command                */
35         {
```

```
36          WORD dac_val ;
37                                          /* Select both VA generators          */
38          SEL_BOTH () ; SEL_VA () ;
39
40          switch (toupper(param1[0]))
41          {
42             case 'X' : dac_val = hex_conv (param1+1) ;
43                        break ;
44             case 'B' : dac_val = bin_conv (param1+1) ;
45                        break ;
46             default  : dac_val = dec_conv (param1  ) ;
47                        break ;
48          }
49
50          if (dac_val == -1)
51             printf ("\x07Invalid parameter!\n\r") ;
52
53          VAGen_WR = dac_val ;
54
55          UNSEL_BOTH () ;
56
57          *param1 = *param2 ;            /* Used to avoid compiler warnings      */
58       }
```

```
59
60
61
62
63          void c_calib_gr (param1, param2)
64          char param1 [] ;                    /* Parameters of command                */
65          char param2 [] ;                    /* Parameters of command                */
66          {
67             WORD dac_val ;
68
69             switch (toupper(param1[0]))
70             {
71                case 'X' : dac_val = hex_conv (param1+1) ;
72                           break ;
73                case 'B' : dac_val = bin_conv (param1+1) ;
74                           break ;
75                default  : dac_val = dec_conv (param1  ) ;
76                           break ;
77             }
78
79             if (dac_val == -1)
80                printf ("\x07Invalid parameter!\n\r") ;
81             else
82             {                                /* Write word to QUAD DAC               */
83                Grid_Volt_WR = 0x4000 | dac_val ;
84                Grid_Volt_WR = 0x5000 | dac_val ;
85                Grid_Volt_WR = 0x6000 | dac_val ;
86                Grid_Volt_WR = 0x7000 | dac_val ;
87             }
88
89             *param1 = *param2 ;              /* Used to avoid compiler warnings      */
90          }
```

```
91
92
93
94          void c_calib_em (param1, param2)
95          char param1 [] ;                    /* Parameters of command                */
96          char param2 [] ;                    /* Parameters of command                */
97          {
98             WORD sel_instr ;
99             REAL va_val    ;
100            WORD dac_val   ;
101            WORD ad_val    ;
102                                             /* Check parameters, select instruments */
103                                             /*     and return instrument value      */
104            sel_instr = proc_instr_params (param1, param2, INSTR_PARAM) ;
105            if (sel_instr == -1)
106               return ;
107                                            /* Make sure instrument is selected      */
108            UNSEL_BOTH () ;
109            IO_PORT1 |= sel_instr ;
110
111            printf ("Electrometer calibration of %x\n\r", sel_instr) ;
112            printf ("Va value    Va DAC word   A/D value\n\r") ;
113
114            for (va_val = -15.0 ; va_val <= 15.0 ; va_val +=0.1)
115            {
116               dac_val = (WORD) (SFM_VA_GAIN * va_val + SFM_VA_OFFSET) ;
117               set_va (sel_instr, va_val) ;
118               delay (5) ;
119               STR_WDOG () ;
120
121               IO_PORT1 |= sel_instr ;
122               START_AD () ;
123               SEL_EM () ;
124               delay (1) ;
125               ad_val = EMeter_RD ;
126
127               shift_oce_vals () ;
128
129               ad_val = ~(EMeter_RD) & 0x0FFF ;
130
131               printf ("%7.3f     0x%4x     %6u\n\r", va_val, dac_val, ad_val) ;
132            }
133         }
```

```
134
135
136
137         void c_calib_vr (param1, param2)
138         char param1 [] ;                    /* Parameters of command              */
139         char param2 [] ;                    /* Parameters of command              */
140         {
141            WORD sel_instr ;
142            WORD vern_val  ;
143            WORD ad_val    ;
144
145                                             /* Check parameters, select instruments */
146                                             /*     and return instrument value      */
147            sel_instr = proc_instr_params (param1, param2, INSTR_PARAM) ;
148            if (sel_instr == -1)
149               return ;
150                                            /* Make sure instrument is selected     */
151            UNSEL_BOTH () ;
152
153            printf ("Vernier calibration of %x\n\r", sel_instr) ;
154            printf ("Vern value      A/D value\n\r") ;
155
156            for (vern_val = 0x000F ; vern_val <= 0x0FFF ; vern_val += 0x0010)
157            {
158               set_vern_pg (sel_instr, vern_val, SET_VERN) ;
159               delay (5) ;
160               STR_WDOG () ;
161
162               IO_PORT1 |= sel_instr ;
163               START_AD () ;
164               SEL_EM () ;
165               delay (1) ;
166               ad_val = EMeter_RD ;
167
168               shift_oce_vals () ;
169
170               ad_val = ~(EMeter_RD) & 0x0FFF ;
171
172               printf ("0x%4x     %6u\n\r", vern_val, ad_val) ;
173            }
174         }
```

```
175
176
177
178         void c_va_sweep (param1, param2)
179         char param1 [] ;                /* Parameters of command              */
180         char param2 [] ;                /* Parameters of command              */
181         {
182            REAL va_val ;
183            char charin ;
184
185            do
186            {
187               for (va_val = -15.0 ; va_val <= +15.0 ; va_val += 0.1)
188               {
189                  set_va (BOTH_INSTR, va_val) ;
190                  STR_WDOG () ;
191               }
192               for (va_val = +15.0 ; va_val >= -15.0 ; va_val -= 0.1)
193               {
194                  set_va (BOTH_INSTR, va_val) ;
195                  STR_WDOG () ;
196               }
197            }
198            while ((Serial_Flags & 0x20) != 0x20) ;
199
200            do
201            {
202               charin = getchar () ;
203            }
204            while (charin != 0x0D) ;
205                                         /* Clear communication port flags     */
206            Clear_Com ();
207
208         *param1 = *param2 ;             /* Used to avoid compiler warnings    */
209         }
```

:ors: none
Warnings: none
Code size: 1081

TYPES.H

```
/*****************************************************************************/
/*                                                                           */
/*     Name:     types.h                                                     */
/*                                                                           */
/*     Params:   None.                                                       */
/*                                                                           */
/*     Purpose:  To provide consistent portable type definitions through    */
/*               typedefs.                                                   */
/*                                                                           */
/*     Mods:     Jan_12_91   NRV   Authored.                                 */
/*                                                                           */
/*****************************************************************************/


#ifndef TYPES_INCLUDED

typedef unsigned char  OCTET   ;  /*  8 bit unsigned integer                 */
typedef unsigned char  BYTE    ;  /*  8 bit unsigned integer                 */
typedef unsigned int   WORD    ;  /* 16 bit unsigned integer                 */
typedef          int   INT     ;  /* 16 bit   signed integer                 */
typedef          float REAL    ;  /* 32 bit floating point number            */
typedef unsigned long  DBLWORD ;  /* 32 bit unsigned integer                 */


#define TRUE              1
#define FALSE             0


#define TYPES_INCLUDED 1

#endif
```

DEBUG.H

```
/*************************************************************************/
/*                                                                       */
/*   Module:     debug.h                                                 */
/*                                                                       */
/*   Purpose:    The purpose of this header file is to define structures */
/*               needed to use the CCSDS packet telemetry standard.      */
/*                                                                       */
/*   Inputs:     None.                                                   */
/*                                                                       */
/*   Outputs:    None.                                                   */
/*                                                                       */
/*   Uses:       Type definitions from "types.h".                        */
/*                                                                       */
/*   History:    06_Feb_92   Written.                                    */
/*                                                                       */
/*************************************************************************/

#include "types.h"

#ifdef MAIN

WORD    D_count      ;          /* Counter for loops   (for DEBUG)      */
WORD    DEBUG_LEVEL  ;          /* Current level of debugging           */

#else

extern WORD    D_count      ;   /* Counter for loops   (for DEBUG)      */
extern WORD    DEBUG_LEVEL  ;   /* Current level of debugging           */

endif

#define DEBUG              0       /* If 0, don't even compile debugging   */

#define DEBUG_INIT         0x0000  /* Initial value of DEBUG_LEVEL         */

#define DEBUG_MODES        0x0008
#define DEBUG_CONVERGENCE  0x0004
#define DEBUG_TM           0x0002
#define DEBUG_IO           0x0001
#define DEBUG_SFM_MODES    0x0080
#define DEBUG_ALP_MODES    0x0040
#define DEBUG_MODES_TM     0x0020
#define DEBUG_LOW_TM       0X0010
#define DEBUG_CONV_ALG     0X0100
```

SFM-ALP.C

```
#################################################################################
#                                                                               #
#     Archimedes 8096/80C196 C-Compiler V3.00I/DOS                14/01/92  16:28:02   #
#                                                                               #
#          Target option =  80C196                                              #
#          Memory model  =  large                                               #
#          Source file   =  sfm-alp.c                                           #
#          List file     =  sfm-alp.lst                                         #
#          Object file   =  sfm-alp.r15                                         #
#          ASM file      =  sfm-alp.s15                                         #
#          Command line  =  sfm-alp.c -ml -vl -z -P -e -g -L -F -p52 -A         #
#                                                                               #
#                                   (c) Copyright Archimedes Software Inc. 1990 #
#################################################################################
```

```
1    /*****************************************************************/
2    /*                                                               */
3    /*     Name:     sfm-alp.c                                       */
4    /*                                                               */
5    /*     Params:   None.                                           */
6    /*                                                               */
7    /*     Input:    Command strings from RS-422 port.               */
8    /*                                                               */
9    /*     Output:   Messages via RS-422 port, NASA telemetry, and   */
10   /*               programmed control of the instrument.           */
11   /*                                                               */
12   /*     Purpose:  To control the Solar Flux Monitor and Advanced  */
13   /*               Langmuir Probe during a rocket flight.          */
14   /*                                                               */
15   /*     Mods:     Dec_12_91   NRV   Authored.                     */
16   /*                                                               */
17   /*****************************************************************/
18
19   #define MAIN     TRUE
20
21   #define VERSION "0.999"            /* Software version release      */
22
23   #include "tm.h"
24   #include <io80c196.h>
25   #include <stdio.h>
26   #include "types.h"
27   #include "debug.h"
28   #include "com_proc.h"
29   #include "modes.h"
30   #include "io.h"
31
32
33   /*****************************************************************
34
35   void init_system () ;
```

```
36
37          void send_hello  () ;
38
39          /*****************************************************************/
40
41          void main ()
42          {
43             BYTE  len          ;          /* Length of command string          */
44             char  command[255] ;          /* String read in from keyboard      */
45
46                                           /* Strobe Watchdog timer             */
47             STR_WDOG () ;
48                                           /* Initialize hardware & flags       */
49             init_system () ;
50                                           /* Clear screen and put up message   */
51             c_clrscr (NULL, NULL) ;
52             send_hello  () ;
53                                           /* Strobe Watchdog timer             */
54             STR_WDOG () ;
55                                           /* Test ROM, ZRAM and Telemetry FIFO */
56             ROM_Test    () ;
57             ZRAM_Test   () ;
58             Telem_Test  () ;
59                                           /* Strobe Watchdog timer             */
60             STR_WDOG () ;
61                                           /* Display various status info       */
62             c_status (NULL, NULL) ;
63                                           /* Display prompt for next command   */
64             printf ("SFM_ALP> ") ;
65                                           /* Repeat forever                    */
66             do
67             {                             /* Go to sleep for rest of 20mS      */
68                sleep () ;
69
70                if ( ACCEL_STAT() > 0 || JIFF_RUN() )
71                {
72                                           /* Update clock value in Zero Power RAM */
73                   Clock_ZP++ ;
74                   T_START () ;
75                                           /* Start A/D Conversion              */
76                   WDOG_HIGH  () ;
77                   SEL_BOTH   () ;
78                   START_AD   () ;
79                   UNSEL_BOTH () ;
80                   delay (1) ;
81                                           /* Place converted value(s) into TM  */
82                                           /*    packet (if appropriate)        */
83                   data_to_tm () ;
84                                           /* Set up next step(s) in mode       */
85                   next_mode_step () ;
```

```
                                        /* Telemeter any complete packets        */
                send_ready_tm () ;

                WDOG_LOW () ;
              }
              else
              {
                if ((idle_count % IDLE_TM_TIME) == 0)
                {
                  IDLE_telem.instr_hdr.start_time = 0x12340000 + idle_count ;
                  IDLE_packet_ready = TRUE ;  /* Send idle packet occasionally   */
                  send_ready_tm () ;
                }
                idle_count++ ;
                STR_WDOG () ;
              }
                                        /* If CR is detected                     */
              if ((Serial_Flags & 0x20) == 0x20)
              {
                                        /* Read command from buffer              */
                len = -1 ;
                do
                  command[++len] = getchar () ;
                while (command[len] != 0x0D) ;
                command[len] = '\0' ;

                                        /* Clear communication port flags        */
                Clear_Com ();
                                        /* Process command string                */
                com_proc (command, len) ;
                                        /* Display prompt for next command       */
                printf ("SFM_ALP> ") ;
              }
            }
            while ( TRUE ) ;

          }
```

```
124
125
126
127         void init_system()
128         {
129                                          /* Initialize debugging flags          */
130            DEBUG_LEVEL = DEBUG_INIT ;
131                                          /* Initialize optocoupled electrometer's */
132                                          /*    clock, preamp & vernier registers  */
133            init_oces () ;
134                                          /* Initialize VA Generators (0 Volts)    */
135                                          /* Unselect both instruments (SFM & ALP) */
136            set_va (BOTH_INSTR, 0.0) ;
137                                          /* Initialize Grid Voltages (0 Volts)    */
138            set_grid (1, 0.0) ;
139            set_grid (2, 0.0) ;
140            set_grid (3, 0.0) ;
141            set_grid (4, 0.0) ;
142                                          /* Initialize Probe Cleaning Control     */
143            Probe_Cln_WR = 0x8000   ;
144            clean_word   = 0x8000   ;
145                                          /* Initialize Telemetry System           */
146            init_telem () ;
147
148            if (ACCEL_STAT () == 0)
149            {
150                                           /* Reinitialize mode pointers           */
151            init_modes () ;
152                                           /* Reset timer back to zero             */
153            Clock_ZP = 0 ;
154            }
155
156         }
```

```
157
158
159
160         void send_hello  ()
161         {
162            printf ("Solar Flux Monitor and Advanced Langmuir Probe\n\r") ;
163            printf ("           Flight Control Software\n\r") ;
164            printf ("-------------------------------------------\n\r") ;
165            printf ("Version %s  Compiled on %s %s\n\r",
166                     VERSION, __DATE__, __TIME__) ;
167            printf ("-------------------------------------------\n\r") ;
168         }
```

169

Errors: none
Warnings: none
Code size: 557

COM-PROC

```
/*************************************************************************/
/*                                                                       */
/*   Module:    com_proc.h                                               */
/*                                                                       */
/*   Purpose:   The purpose of this header file is to define the command */
/*              processor structures.                                    */
/*                                                                       */
/*   Inputs:    None.                                                    */
/*                                                                       */
/*   Outputs:   None.                                                    */
/*                                                                       */
/*   Uses:      Nothing from other files.                                */
/*                                                                       */
/*   History:   14_Feb_92   Written.                                     */
/*                                                                       */
/*************************************************************************/

#include "types.h"

/*************************************************************************/

#define CMD_ELEM            10    /* Number of elements in commmand line  */
#define CMD_LEN             10    /* Max Length of command line element   */

typedef struct cmdstruct  {
   char   *command    ;            /* Command string (in uppercase         */
   BYTE   minlen      ;            /* Min # of chars needed for command    */
   void  (*cmd_fun) () ;           /* Point to processing subroutine       */
   BYTE   num_params  ;            /* Number of parameters for command     */
  cmd_array ;

/*************************************************************************/

#define REAL_PARAM    99
#define WORD_PARAM    55
#define GRID_PARAM    33
#define TELEM_PARAM   22
#define INSTR_PARAM   00


#ifdef COM_PROC_C

void com_proc   (char [], int len) ;

void c_help     (char [], char []) ;
void c_reset    (char [], char []) ;
void c_tstart   (char [], char []) ;
void c_tstop    (char [], char []) ;
void c_smode    (char [], char []) ;
void c_status   (char [], char []) ;
void c_ser_tm   (char [], char []) ;

void c_set_va   (char [], char []) ;
void c_set_grid (char [], char []) ;
void c_set_vern (char [], char []) ;
void c_set_pg   (char [], char []) ;
void c_sample   (char [], char []) ;
void c_relay    (char [], char []) ;
void c_tm_test  (char [], char []) ;
```

```
void c_convtest (char [], char []) ;
void c_convtest2(char [], char []) ;

void c_clrscr   (char [], char []) ;
void c_debug    (char [], char []) ;
/*
void c_setport  (char [], char []) ;
void c_w_299    (char [], char []) ;
void c_r_299    (char [], char []) ;
void c_shift    (char [], char []) ;
void c_start_ad (char [], char []) ;
void c_latch_gn (char [], char []) ;
*/
void c_loader   (char [], char []) ;

BYTE proc_instr_params (char [], char [], BYTE param_type) ;

void uppercase  (char *strptr) ;

WORD hex_conv   (char param[]) ;
WORD dec_conv   (char param[]) ;
WORD bin_conv   (char param[]) ;

char  comm_v[CMD_ELEM][CMD_LEN] ;  /* Array of command elements          */
WORD  comm_c                    ;  /* Number of elements in array        */

const cmd_array commands[] = {  { "HELP"     ,  1, c_help    , 0 },
                                { "?"        ,  1, c_help    , 0 },
                                { "RESET"    ,  5, c_reset   , 0 },
                                { "TSTART"   ,  4, c_tstart  , 0 },
                                { "TSTOP"    ,  4, c_tstop   , 0 },
                                { "SMODE"    ,  4, c_smode   , 1 },
                                { "STATUS"   ,  3, c_status  , 0 },
                                { "SERTM"    ,  3, c_ser_tm  , 1 },

                                { "VA"       ,  2, c_set_va  , 2 },
                                { "GRID"     ,  2, c_set_grid, 2 },
                                { "VERN"     ,  4, c_set_vern, 2 },
                                { "PG"       ,  2, c_set_pg  , 2 },
                                { "SAMPLE"   ,  2, c_sample  , 0 },
                                { "RELAY"    ,  3, c_relay   , 1 },
                                { "CALIBVA"  ,  7, c_calib_va, 1 },
                                { "CALIBGR"  ,  7, c_calib_gr, 1 },
                                { "CALIBEM"  ,  7, c_calib_em, 1 },
                                { "CALIBVERN",  9, c_calib_vr, 1 },
                                { "CONVTEST" ,  8, c_convtest, 2 },
                                { "CONVTEST2",  9, c_convtest2,2 },
                                { "VASWEEP"  ,  7, c_va_sweep, 0 },
                                { "TMTEST"   ,  6, c_tm_test , 0 },

                                { "CLS"      ,  3, c_clrscr  , 0 },
                                { "DEBUG"    ,  5, c_debug   , 1 },
/*                                { "PORT"     ,  2, c_setport , 1 },
                                { "W299"     ,  2, c_w_299   , 1 },
                                { "R299"     ,  2, c_r_299   , 0 },
                                { "SHIFT"    ,  2, c_shift   , 0 },
                                { "START"    ,  2, c_start_ad, 0 },
                                { "LATCH"    ,  2, c_latch_gn, 0 },
*/
                                { "LOADER"   ,  6, c_loader  , 0 }   } ;
```

```
#else

extern void com_proc   (char [], int len) ;

extern void c_status   (char [], char []) ;

extern void c_clrscr   (char [], char []) ;

extern BYTE proc_instr_params (char [], char [], BYTE param_type) ;

extern void uppercase  (char *strptr) ;

extern WORD hex_conv   (char param[]) ;
extern WORD dec_conv   (char param[]) ;
extern WORD bin_conv   (char param[]) ;

#endif

/*****************************************************************************/

extern void     loader    () ;
extern void     Clear_Com () ;

#define NUM_COMMANDS  (sizeof (commands) / sizeof(cmd_array))

#define STRNCOMP(str1,str2,n)     (strncmp(str1,str2,n) == 0)
#define STRCOMP(str1,str2)        (strcmp(str1,str2)    == 0)
#define MAX(a,b)                  (a > b ?  a : b)
```

```
#################################################################################
#                                                                               #
#    Archimedes 8096/80C196 C-Compiler V3.00I/DOS                14/Jul/92  16:23:59   #
#                                                                               #
#         Target option =  80C196                                               #
#         Memory model  =  large                                                #
#         Source file   =  com_proc.c                                           #
#         List file     =  com_proc.lst                                         #
#         Object file   =  com_proc.r15                                         #
#         ASM file      =  com_proc.s15                                         #
#         Command line  =  com_proc.c -ml -vl -z -P -e -g -L -F -p52 -A         #
#                                                                               #
#                                   (c) Copyright Archimedes Software Inc. 1990 #
#################################################################################
```

```
1     /*************************************************************************/
2     /*                                                                       */
3     /*     Name:     com_proc.c                                              */
4     /*                                                                       */
5     /*     Params:   None.                                                   */
6     /*                                                                       */
7     /*     Input:    None.                                                   */
8     /*                                                                       */
9     /*     Output:                                                           */
10    /*                                                                       */
11    /*                                                                       */
12    /*     Purpose:                                                          */
13    /*                                                                       */
14    /*                                                                       */
15    /*     Mods:     Jan_12_91   NRV   Authored.                             */
16    /*                                                                       */
17    /*************************************************************************/
18
19    #define COM_PROC_C TRUE
20
21    #include <stdio.h>
22    #include <stdlib.h>
23    #include <ctype.h>
24    #include <string.h>
25    #include <io80c196.h>
26    #include "types.h"
27    #include "debug.h"
28    #include "calib.h"
29    #include "com_proc.h"
30    #include "modes.h"
31    #include "io.h"
32    #include "tm.h"
33
34
35    REAL real_out ;
```

```
36          WORD word_out ;
37
38          void com_proc (char command [], int len)
39          {
40             BYTE  pos          ;          /* Position in command string          */
41             BYTE  cpos         ;          /* Position in current command element  */
42             char  prev         ;          /* Previous character                  */
43             BYTE  command_OK   ;          /* Is it a valid command               */
44
45                                           /* PARSE (arghhhhh!!!) command line and */
46                                           /*   stuff space delimited words into   */
47                                           /*   an argv type string array!!!       */
48             comm_c =   0 ;
49             cpos   =   0 ;
50             pos    =   0 ;
51             prev   = ' ' ;
52             do
53             {
54                if (isspace (command[pos]))
55                {
56                   if (!isspace(prev))
57                   {
58                      comm_v[comm_c][cpos] = '\0' ;
59                      comm_c++ ;
60                      cpos = 0 ;
61                      prev = command[pos] ;
62                   }
63                }
64                else
65                {
66                   comm_v[comm_c][cpos] = command[pos] ;
67                   prev = command[pos] ;
68                   cpos++ ;
69                }
70                pos++ ;
71             }
72             while ((pos <= len) && (comm_c < CMD_ELEM)) ;
73
74             if (len != 0)
75                comm_c++ ;
76                                           /* Convert comm_c[0] to uppercase      */
77             uppercase (comm_v[0]) ;
78
79                                           /* Process command                     */
80             if (comm_c != 0)
81             {
82                command_OK = FALSE ;
83                pos = 0 ;
84                do
85                {
```

```
86              if ( STRNCOMP(comm_v[0], commands[pos].command,
87                          MAX(commands[pos].minlen,strlen(comm_v[0]))))
88              {
89                 if ((comm_c - 1 ) == commands[pos].num_params)
90                    (*commands[pos].cmd_fun) (comm_v[1],comm_v[2]) ;
91                 else
92                    printf ("\007Invalid number of parameters\n\r") ;
93                 command_OK = TRUE ;
94              }
95              pos++ ;
96           }
97           while ((pos < NUM_COMMANDS) && (!command_OK)) ;
98           if (!command_OK)
99              printf ("\007Unrecognized command !!\n\r") ;
100        }
101     }
```

```
/****************************************************************************/
/****************************************************************************/
/*      Command processing functions start here!!!!!!!!!!!!!!!!!!!!      */
/****************************************************************************/
/****************************************************************************/

void c_help     (param1, param2)
char param1 [] ;                  /* Parameters of command              */
char param2 [] ;                  /* Parameters of command              */
{
   c_clrscr (NULL, NULL) ;        /* Clear screen                       */

printf ("\n\r") ;
printf ("HELP or ?           Display this help screen                \n\r") ;
printf ("RESET               Ready CPU values for flight (clock, etc.) \n\r") ;
printf ("TSTART              Manually start timer and simulate launch  \n\r") ;
printf ("TSTOP               Stop manually started timer             \n\r") ;
printf ("TSET         val    Set timer value to 'val'                \n\r") ;
printf ("SMODE  instr mode   Run specified instrument mode           \n\r") ;
printf ("STATUS              Display system status                   \n\r") ;
printf ("SERTM  ON/OFF       Set serial telemetry on or off          \n\r") ;

printf ("GRID   grid# val    Set grid voltage (1-3) DAC (12 bits)    \n\r") ;
printf ("VA     instr val    Set 'instr's VA Gen DAC (16 bits)       \n\r") ;
printf ("VERN   instr val    Set electrometer's vernier (12 bits)    \n\r") ;
printf ("PG     instr val    Set electrometer's preamp gain (3 bits) \n\r") ;
printf ("SAMPLE instr        Get sample from instrument's A/D (12 bits)\n\r") ;
printf ("RELAY        val    Write 'val' to relay control latch      \n\r") ;

printf ("LOADER              Jump to Loader program                  \n\r") ;

printf ("\n\r") ;

*param1 = *param2 ;               /* Used to avoid compiler warnings    */
}
```

```
140
141
142
143        void c_reset    (param1, param2)
144        char param1 [] ;                /* Parameters of command              */
145        char param2 [] ;                /* Parameters of command              */
146        {
147                                        /* Stop timer override                */
148           T_STOP () ;
149                                        /* Reinitialize mode pointers         */
150           init_modes () ;
151                                        /* Reset timer back to zero           */
152           Clock_ZP = 0 ;
153        *param1 = *param2 ;             /* Used to avoid compiler warnings    */
154        }
```

```
155
156
157
158         void c_tstart   (param1, param2)
159         char param1 [] ;                /* Parameters of command              */
160         char param2 [] ;                /* Parameters of command              */
161         {
162                                         /* Enable Jiffy Override Flag         */
163            T_START () ;
164
165         *param1 = *param2 ;             /* Used to avoid compiler warnings    */
166         }
```

```
167
168
169
170         void c_tstop    (param1, param2)
171         char param1 [] ;                /* Parameters of command              */
172         char param2 [] ;                /* Parameters of command              */
173         {
174                                         /* Disable Jiffy Override Flag        */
175            T_STOP () ;
176
177         *param1 = *param2 ;             /* Used to avoid compiler warnings    */
178         }
```

```
179
180
181
182     void c_smode  (param1, param2)
183     char param1 [] ;
184     char param2 [] ;                /* Parameters of command          */
185     {                               /* Parameters of command          */
186
187     *param1 = *param2 ;
188     }                               /* Used to avoid compiler warnings */
```

```
189
190
191
192         void c_status (param1, param2)
193         char param1 [] ;                    /* Parameters of command              */
194         char param2 [] ;                    /* Parameters of command              */
195         {
196            int ad_val_in ;
197
198            printf ("-------------------------------------------\n\r") ;
199            printf ("Launch Detect = %s\n\r",
200                      (ACCEL_STAT() > 0 ? "TRUE" : "FALSE") ) ;
201            printf ("Time  = %7.2f s = %lu\n\r", (REAL) Clock_ZP / TICKS_PER_SECOND,
202                                                 Clock_ZP   ) ;
203            printf ("Serial TM is %s\n\r", (send_serial_tm > 0 ? "ON" : "OFF")) ;
204            printf ("Mode %u = %s @ step %u\n\r", SFM_Mode_Ptr,
205                      modelist [Curr_SFM_Mode].sname, SFM_Mode_Step) ;
206            printf ("Mode %u = %s @ step %u\n\r", ALP_Mode_Ptr,
207                      modelist [Curr_ALP_Mode].sname, ALP_Mode_Step) ;
208            printf ("Flags = 0x%02X  Status = 0x%02X\n\r", Flags, Status) ;
209            printf ("Port1 = 0x%02X  Port2  = 0x%02X\n\r", IO_PORT1, IO_PORT2) ;
210            printf ("SFM is %s\n\r",
211                      (MON_SFM_RLY() > 0 ? "MEASURING" : "CLEANING") ) ;
212            printf ("ALP is %s\n\r",
213                      (MON_ALP_RLY() > 0 ? "MEASURING" : "CLEANING") ) ;
214            ad_val_in = read_AD (3) ;
215            printf ("+28V mon = %5d\n\r", ad_val_in) ;
216            printf ("-------------------------------------------\n\n\r") ;
217
218         *param1 = *param2 ;                 /* Used to avoid compiler warnings    */
219         }
```

```
220
221
222
223         void c_ser_tm (param1, param2)
224         char param1 [] ;                    /* Parameters of command              */
225         char param2 [] ;                    /* Parameters of command              */
226         {
227            uppercase (param1) ;
228
229            if      ( STRNCOMP(param1, "ON" , strlen(param1)))
230               send_serial_tm = TRUE  ;
231            else if ( STRNCOMP(param1, "OFF", strlen(param1)))
232               send_serial_tm = FALSE ;
233            else
234               printf ("\x07Invalid parameter!\n\r") ;
235
236         *param1 = *param2 ;                 /* Used to avoid compiler warnings    */
237         }
```

```
238
239
240
241
242         /*************************************************************************/
243         /*************************************************************************/
244
245         void c_set_va   (param1, param2)
246         char param1 [] ;                  /* Parameters of command               */
247         char param2 [] ;                  /* Parameters of command               */
248         {
249            BYTE sel_instr ;
250                                           /* Check parameters, select instruments */
251                                           /*     and return instrument value      */
252            sel_instr = proc_instr_params (param1, param2, REAL_PARAM) ;
253            if (sel_instr == -1)
254               return ;
255
256            set_va (sel_instr, real_out) ;  /* Write value to latches & D/A        */
257
258            UNSEL_BOTH() ;                 /* Unselect both instruments           */
259         }
```

```
260
261
262
263         void c_set_grid (param1, param2)
264         char param1 [] ;                    /* Parameters of command              */
265         char param2 [] ;                    /* Parameters of command              */
266         {
267            BYTE grid_num ;
268                                             /* Check parameters, select instruments */
269                                             /*     and return instrument value     */
270            grid_num = (BYTE) proc_instr_params (param1, param2, GRID_PARAM) ;
271            if (grid_num == -1)
272               return ;
273
274            set_grid (grid_num, real_out) ;   /* Write value to D/A converter        */
275         }
```

```
276
277
278
279         void c_set_vern (param1, param2)
280         char param1 [] ;                /* Parameters of command              */
281         char param2 [] ;                /* Parameters of command              */
282         {
283            BYTE sel_instr ;
284                                         /* Check parameters, select instruments */
285                                         /*     and return instrument value     */
286            sel_instr = proc_instr_params (param1, param2, WORD_PARAM) ;
287            if (word_out == -1)
288               return ;
289                                         /* Not & Truncate vernier value       */
290            word_out = word_out & 0x0FFF ;
291
292            set_vern_pg (sel_instr, word_out, SET_VERN) ;
293         }
```

```
294
295
296
297         void c_set_pg   (param1, param2)
298         char param1 [] ;                /* Parameters of command              */
299         char param2 [] ;                /* Parameters of command              */
300         {
301            BYTE sel_instr ;
302                                         /* Check parameters, select instruments */
303                                         /*     and return instrument value     */
304            sel_instr = proc_instr_params (param1, param2,WORD_PARAM) ;
305            if (word_out == -1)
306               return ;
307
308            word_out  &= 0x000F ;        /* Truncate preamp gain value to 4-bits */
309            word_out  <<= 12    ;        /* Shift 4-bits to MS nibble          */
310
311            set_vern_pg (sel_instr, word_out, SET_PG) ;
312         }
```

```
313
314
315
316         void c_sample   (param1, param2)
317         char param1 [] ;                    /* Parameters of command              */
318         char param2 [] ;                    /* Parameters of command              */
319         {
320            WORD trash ;
321
322            SEL_BOTH () ;
323
324            START_AD () ;
325
326            SEL_EM () ;
327
328            delay (1) ;
329
330            trash = EMeter_RD ;
331
332            shift_oce_vals () ;
333
334            UNSEL_SFM () ;
335            new_ALP_AD_val = EMeter_RD ;
336            new_ALP_AD_val = ~(EMeter_RD) & 0x0FFF ;
337
338            UNSEL_ALP () ;
339            SEL_SFM () ;
340            new_SFM_AD_val = EMeter_RD ;
341            new_SFM_AD_val = ~(EMeter_RD) & 0x0FFF ;
342
343            printf ("ALP A/D value=%d (0x%x)\n\r", new_ALP_AD_val, new_ALP_AD_val) ;
344            printf ("SFM A/D value=%d (0x%x)\n\r", new_SFM_AD_val, new_SFM_AD_val) ;
345
346         *param1 = *param2 ;                 /* Used to avoid compiler warnings     */
347         }
```

```
348
349
350
351        void c_tm_test  (param1, param2)
352        char param1 [] ;                /* Parameters of command              */
353        char param2 [] ;                /* Parameters of command              */
354        {
355           test_telem () ;
356
357        *param1 = *param2 ;             /* Used to avoid compiler warnings    */
358        }
```

```
359
360
361
362         void c_relay    (param1, param2)
363         char param1 [] ;                /* Parameters of command              */
364         char param2 [] ;                /* Parameters of command              */
365         {
366            char new_r_val ;             /* New value for relay control word     */
367
368            switch (toupper(param1[0]))
369            {
370               case 'X' : new_r_val = hex_conv (param1+1) ;
371                          break ;
372               case 'B' : new_r_val = bin_conv (param1+1) ;
373                          break ;
374               default  : new_r_val = dec_conv (param1  ) ;
375                          break ;
376            }
377
378            if (new_r_val == -1)
379               printf ("\x07Invalid parameter!\n\r") ;
380            else
381               Timer_WR = ((WORD) new_r_val & 0x00FF) + 0x8000 ;
382
383            *param1 = *param2 ;          /* Used to avoid compiler warnings     */
384         }
```

```
385
386
387
388         void c_convtest (param1, param2)
389         char param1 [] ;                    /* Parameters of command                */
390         char param2 [] ;                    /* Parameters of command                */
391         {
392            BYTE step ;
393
394            BYTE sel_instr ;
395                                             /* Check parameters, select instruments */
396                                             /*     and return instrument value      */
397            sel_instr = proc_instr_params (param1, param2,WORD_PARAM) ;
398            if (word_out == -1)
399               return ;
400            if (sel_instr == -1)
401               return ;
402
403            for (step = 0 ; step < NUM_PVG_SET_STEPS ; step++)
404            {                                 /* Start A/D Conversion                 */
405               STR_WDOG   () ;
406               SEL_BOTH   () ;
407               START_AD   () ;
408               UNSEL_BOTH () ;
409               delay (1) ;
410                                             /* Place converted value(s) into TM     */
411                                             /*    packet (if appropriate)           */
412               data_to_tm () ;
413
414               set_mode_pvg (step, sel_instr, word_out) ;
415               delay (4) ;
416            }
417
418            if (sel_instr == ALP_INSTR)
419               printf ("ALP pg=%d vern=0x%X\n\r", (ALP_Vern_PG & 0xf000)>>12,
420                                                ~(ALP_Vern_PG) & 0x0FFF ) ;
421            else
422               printf ("SFM pg=%d vern=0x%X\n\r", (SFM_Vern_PG & 0xf000)>>12,
423                                                ~(SFM_Vern_PG) & 0x0FFF ) ;
424
425            *param1 = *param2 ;              /* Used to avoid compiler warnings      */
426         }
```

```
427
428
429
430         void c_convtest2 (param1, param2)
431         char param1 [] ;                    /* Parameters of command              */
432         char param2 [] ;                    /* Parameters of command              */
433         {
434            BYTE step      ;
435            REAL va_val    ;
436            BYTE sel_instr ;
437                                             /* Check parameters, select instruments */
438                                             /*     and return instrument value      */
439            sel_instr = proc_instr_params (param1, param2,WORD_PARAM) ;
440            if (word_out == -1)
441               return ;
442            if (sel_instr == -1)
443               return ;
444
445            for (step = 0 ; step < NUM_VCONVERGE_STEPS ; step++)
446            {                                 /* Start A/D Conversion               */
447               STR_WDOG   () ;
448               SEL_BOTH   () ;
449               START_AD   () ;
450               UNSEL_BOTH () ;
451               delay (1) ;
452                                             /* Place converted value(s) into TM     */
453                                             /*    packet (if appropriate)           */
454               data_to_tm () ;
455
456               vconverge (step, sel_instr, word_out) ;
457               delay (4) ;
458            }
459
460            switch (sel_instr)
461            {
462               case  SFM_INSTR :
463                        va_val = ((REAL) va_conv_val - SFM_VA_OFFSET) / SFM_VA_GAIN ;
464                        break ;
465               case  ALP_INSTR :
466                        va_val = ((REAL) va_conv_val - ALP_VA_OFFSET) / ALP_VA_GAIN ;
467                        break ;
468            }
469
470            if (sel_instr = ALP_INSTR)
471               printf ("ALP va=%7.3f (0x%X)  ad=0x%X\n\r",
472                        va_val, va_conv_val, new_ALP_AD_val) ;
473            else
474               printf ("SFM va=%7.3f (0x%X)  ad=0x%X\n\r",
475                        va_val, va_conv_val, new_SFM_AD_val) ;
476
```

```
477             *param1 = *param2 ;            /* Used to avoid compiler warnings      */
478         }
```

```
479
480
481
482     /*************************************************************************/
483     /*************************************************************************/
484
485
486     void c_clrscr   (param1, param2)
487     char param1 [] ;                /* Parameters of command              */
488     char param2 [] ;                /* Parameters of command              */
489     {
490        printf ("\f") ;
491
492     *param1 = *param2 ;             /* Used to avoid compiler warnings    */
493     }
```

```
494
495
496
497         void c_debug    (param1, param2)
498         char param1 [] ;                /* Parameters of command              */
499         char param2 [] ;                /* Parameters of command              */
500         {
501            WORD new_d_lev ;             /* New level of debugging             */
502
503            switch (toupper(param1[0]))
504            {
505               case 'X' : new_d_lev = hex_conv (param1+1) ;
506                          break ;
507               case 'B' : new_d_lev = bin_conv (param1+1) ;
508                          break ;
509               default  : new_d_lev = dec_conv (param1  ) ;
510                          break ;
511            }
512
513            if (new_d_lev == -1)
514               printf ("\x07Invalid parameter!\n\r") ;
515            else
516            {
517               printf ("Debugging Level set to %d(0x%x)\n\r", new_d_lev) ;
518               DEBUG_LEVEL = (unsigned char) new_d_lev ;
519            }
520
521            *param2 = *param1 ;          /* Used to avoid compiler warnings    */
522         }
```

```
523
524
525
526        void c_loader   (param1, param2)
527        char param1 [] ;                /* Parameters of command              */
528        char param2 [] ;                /* Parameters of command              */
529        {
530           loader () ;
531
532           *param1 = *param2 ;             /* Used to avoid compiler warnings      */
533        }
```

```
/***************************************************************************/
/***************************************************************************/
/*              Support functions start here!!!!!!!!!!!!!!!!!!!            */
/***************************************************************************/
/***************************************************************************/


BYTE proc_instr_params (param1, param2, param_type)
char param1 [] ;                  /* Parameters of command                 */
char param2 [] ;                  /* Parameters of command                 */
BYTE param_type ;                 /* Type of parameter                     */
{
   BYTE ret_val ;

   uppercase (param1) ;

   if (param_type != GRID_PARAM)
   {
      if      ( STRNCOMP(param1, "SFM" , strlen(param1)))
         ret_val = SFM_INSTR ;
      else if ( STRNCOMP(param1, "ALP" , strlen(param1)))
         ret_val = ALP_INSTR ;
      else if ( STRNCOMP(param1, "BOTH", strlen(param1)))
         ret_val = BOTH_INSTR ;
      else
      {
         printf ("\x07Invalid parameter \'%s\'!\n\r",param1) ;
         return (-1) ;
      }
   }
   else
      ret_val = dec_conv (param1 ) ;

   switch (param_type)
   {
      case GRID_PARAM :
      case REAL_PARAM :
         real_out = 0.0 ;
         sscanf (param2, "%f", &real_out) ;
         break ;
      case INSTR_PARAM :
         break ;
      case WORD_PARAM :
      default         :
         if      (toupper(param2[0]) == 'X')
            word_out = hex_conv (param2+1) ;
         else if (toupper(param2[0]) == 'B')
```

```
584                     word_out = bin_conv (param2+1) ;
585                  else
586                     word_out = dec_conv (param2  ) ;
587
588                  if (word_out == -1)
589                  {
590                     printf ("\x07Invalid parameter \'%s\'!\n\r", param2) ;
591                     return (-1) ;
592                  }
593                  break ;
594               }
595
596               return (ret_val) ;
597            }
```

```
598
599
600
601        void uppercase (char *strptr)      /* Convert string to uppercase          */
602        {
603           WORD pos ;
604
605           pos = strlen (strptr) ;
606           do
607              strptr[pos] = toupper (strptr[pos]) ;
608           while (pos-- > 0) ;
609        }
```

```
610
611
612
613         WORD hex_conv (param)
614         char param [] ;
615         {
616            BYTE plen       ;              /* Length of parameter string          */
617            BYTE count      ;              /* Counter for loop                    */
618            char c          ;              /* Uppercase version of current char   */
619            WORD outval = 0 ;              /* Value to be returned                */
620
621                                           /* Get length of string to be converted */
622            plen = strlen (param) ;
623                                           /* Loop character by character         */
624            for (count = 0 ; count < plen ; count++)
625            {
626               c = toupper (param[count]) ;
627               if     (isdigit  (c))
628                  outval = (outval << 4) + (c - '0'     ) ;
629               else if (c >= 'A' && c <= 'F')
630                  outval = (outval << 4) + (c - 'A' + 10) ;
631               else
632                  return (-1) ;
633            }
634
635            return (outval) ;
636         }
```

```
637
638
639
640         WORD dec_conv (param)
641         char param [] ;
642         {
643            BYTE  plen      ;              /* Length of parameter string          */
644            BYTE  count     ;              /* Counter for loop                    */
645            char c          ;              /* Uppercase version of current char   */
646            WORD outval = 0 ;              /* Value to be returned                */
647
648                                           /* Get length of string to be converted */
649            plen = strlen (param) ;
650                                           /* Loop character by character         */
651            for (count = 0 ; count < plen ; count++)
652            {
653               c = toupper (param[count]) ;
654               if     (isdigit  (c))
655                  outval = (outval * 10) + (c - '0') ;
656               else
657                  return (-1) ;
658            }
659
660            return (outval) ;
661         }
```

```
662
663
664
665         WORD bin_conv (param)
666         char param [] ;
667         {
668            BYTE plen       ;              /* Length of parameter string          */
669            BYTE count      ;              /* Counter for loop                    */
670            char c          ;              /* Uppercase version of current char   */
671            WORD outval = 0 ;              /* Value to be returned                */
672
673                                           /* Get length of string to be converted */
674            plen = strlen (param) ;
675                                           /* Loop character by character         */
676            for (count = 0 ; count < plen ; count++)
677            {
678               c = toupper (param[count]) ;
679               if (c == '1' || c == '0')
680                  outval = (outval << 1) + (c - '0') ;
681               else
682                  return (-1) ;
683            }
684
685            return (outval) ;
686         }
```

```
rrors: none
arnings: none
Code size: 3690
```

MODES

```
364
365
366
367         void alp_hr_ne ()
368         {
369            if (ALP_Mode_Step == 0)
370            {
371               set_va (ALP_INSTR, 5.0) ;
372            }
373
374            if (ALP_Mode_Step < NUM_PVG_SET_STEPS)
375            {                              /* Set preamp & vernier gain near FS     */
376               set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_50_FS) ;
377            }
378         }
```

```
379
380
381
382         void alp_clean  ()
383         {
384            if (ALP_Mode_Step == 0)
385            {
386               set_va   (ALP_INSTR,   0.0) ;
387            }
388                                          /* Perform switching to clean probe      */
389            if (ALP_Mode_Step <= 2)
390               clean_prb (ALP_INSTR, ALP_Mode_Step) ;
391                                          /* Perform switching to do measurements  */
392            if (ALP_Mode_Step >= CLEANING_TIME+2)
393               uncln_prb (ALP_INSTR, ALP_Mode_Step - (CLEANING_TIME+2)) ;
394
395            if (ALP_Mode_Step >= (modelist[Curr_ALP_Mode].num_steps - 1))
396            {
397               ALP_packet_ready = TRUE ;
398            }
399         }
```

```
Errors: none
Warnings: none
Code size: 2782
```

IO

```
/*****************************************************************************/
/*                                                                           */
/*   Module:     io.h                                                        */
/*                                                                           */
/*   Purpose:    The purpose of this header file is to define macros and     */
/*               reference routines used for hardware control and I/O.       */
/*                                                                           */
/*   Inputs:     None.                                                       */
/*                                                                           */
/*   Outputs:    None.                                                       */
/*                                                                           */
/*   Uses:       80C196 specific definitions from including <io80C196.h>.    */
/*                                                                           */
/*   History:    06_Feb_92   Written.                                        */
/*                                                                           */
/*****************************************************************************/


#include "types.h"

/*****************************************************************************/

#define ALP_INSTR       0x20
#define SFM_INSTR       0x40
#define BOTH_INSTR      0x60

#define STR_WDOG()    {IO_PORT1 |= 0x80 ; IO_PORT1 &= 0x7F ;} /* Pulse P1.7 */
#define WDOG_HIGH()   {IO_PORT1 |= 0x80 ;} /* Pin P1.7 = 1 */
#define WDOG_LOW()    {IO_PORT1 &= 0x7F ;} /* Pin P1.7 = 0 */

#define SEL_SFM()     {IO_PORT1 |= SFM_INSTR  ;} /* Pin P1.6 = 1 */
#define UNSEL_SFM()   {IO_PORT1 &= 0xBF       ;} /* Pin P1.6 = 0 */

#define SEL_ALP()     {IO_PORT1 |= ALP_INSTR  ;} /* Pin P1.5 = 1 */
#define UNSEL_ALP()   {IO_PORT1 &= 0xDF       ;} /* Pin P1.5 = 0 */

#define SEL_BOTH()    {IO_PORT1 |= BOTH_INSTR ;} /* Pin P1.6 = 1 & P1.5 = 1 */
#define UNSEL_BOTH()  {IO_PORT1 &= 0x9F       ;} /* Pin P1.6 = 0 & P1.5 = 0 */

#define SEL_EM()      {IO_PORT1 |= 0x10       ;} /* Pin P1.4 = 1 */
#define SEL_VA()      {IO_PORT1 &= 0xEF       ;} /* Pin P1.4 = 0 */

#define START_AD()    {IO_PORT1 |= 0x08 ; IO_PORT1 &= 0xF7 ;} /* Pulse P1.3 */
#define START_AD_ON()  {IO_PORT1 |= 0x08 ; }
#define START_AD_OFF() {IO_PORT1 &= 0xF7 ; }

#define LATCH_PVG()   {IO_PORT1 |= 0x04 ; IO_PORT1 &= 0xFB ;} /* Pulse P1.2 */

#define MON_SFM_RLY()  (IO_PORT2 & 0x80)   /* Port pin P2.7 */
#define MON_ALP_RLY()  (IO_PORT2 & 0x40)   /* Port pin P2.6 */

#define SFM_CLEAN_ON()   { IO_PORT1 |= 0x01 ; } /* Port pin P1.0 */
#define ALP_CLEAN_ON()   { IO_PORT1 |= 0x02 ; } /* Port pin P1.1 */
#define SFM_CLEAN_OFF()  { IO_PORT1 &= 0xFE ; }
#define ALP_CLEAN_OFF()  { IO_PORT1 &= 0xFD ; }

#define ACCEL_STAT()   (IO_PORT2 & 0x08)
#define IS_TM_FULL()   (IO_PORT2 & 0x10)

#define T_START()      { Flags |= 0x02  ; }
```

```
#define T_STOP()        { Flags &= 0xFD  ; }
#define JIFF_STOP()     ((Flags &  0x02) = 0)
#define JIFF_RUN()      ((Flags &  0x02) > 0)

#define SFM_CLN_1    0x0001
#define SFM_CLN_2    0x0002
#define SFM_PRB_1    0x0004
#define SFM_PRB_2    0x0008
#define ALP_CLN_1    0x0010
#define ALP_CLN_2    0x0020
#define ALP_PRB_1    0x0040
#define ALP_PRB_2    0x0080

#define SET_VERN     0x42        /* Used in conjunction with set_vern_pg */
#define SET_PG       0x24
#define SET_VERN_PG  0x66

#define PRE_1        0x7000      /* Highest gain range                   */
#define PRE_2        0x3000
#define PRE_3        0x1000
#define PRE_4        0x0000      /* Lowest gain range                    */

/*****************************************************************************/

                                 /* References to variables defined to   */
                                 /*  be external addresses in SFMALP.ASM */
extern WORD  Grid_Volt_WR ;
extern WORD  Timer_WR     ;
extern WORD  Probe_Cln_WR ;
extern WORD  VAGen_WR     ;
extern WORD  EMeter_WR    ;
extern WORD  EMeter_RD    ;
extern WORD  Telem_WR     ;

                                 /* References to variables defined to   */
                                 /*    be in CPU's RAM in SFMALP.ASM     */
extern BYTE  Serial_Flags ;
extern BYTE  Flags        ;
extern BYTE  Flags2       ;
                                 /* References to variables defined to   */
                                 /*    be in ZRAM in SFMALP.ASM          */

extern WORD  SFM_Vern_PG  ;
extern WORD  ALP_Vern_PG  ;

extern WORD  SFM_VA_Val   ;
extern WORD  ALP_VA_Val   ;

extern WORD  clean_word   ;

extern DBLWORD  Clock_ZP  ;

extern BYTE  Status       ;

extern void loader      (        ) ;
extern void delay       (int ms  ) ;
extern void sleep       (        ) ;
extern  int read_AD     (int chan) ;

extern void ROM_Test    (        ) ;
```

```
extern void ZRAM_Test    (          ) ;
extern void Telem_Test   (          ) ;

 !fndef IO_C

extern void init_oces       (                                  ) ;

extern void shift_oce_vals (                                   ) ;
extern void set_grid        (BYTE grid_no, REAL voltage) ;
extern void set_grid_dac    (BYTE grid_no, WORD dac_in ) ;
extern void set_dac4        (              WORD dac_in ) ;
extern void set_va          (BYTE instr  , REAL voltage) ;
extern void set_va_dac      (BYTE instr  , WORD dac_val) ;
extern void set_vern_pg     (BYTE instr  , WORD value, BYTE val_type) ;
extern void clean_prb       (BYTE instr  , BYTE step   ) ;
extern void uncln_prb       (BYTE instr  , BYTE step   ) ;

#else

void init_oces       (                                  ) ;

void shift_oce_vals (                                   ) ;
void set_grid        (BYTE grid_no, REAL voltage) ;
void set_grid_dac    (BYTE grid_no, WORD dac_in ) ;
void set_dac4        (              WORD dac_in ) ;
void set_va          (BYTE instr  , REAL voltage) ;
void set_va_dac      (BYTE instr  , WORD dac_val) ;
void set_vern_pg     (BYTE instr  , WORD value, BYTE val_type) ;
void clean_prb       (BYTE instr  , BYTE step   ) ;
 oid uncln_prb       (BYTE instr  , BYTE step   ) ;
.ORD read_ad         (BYTE instr                ) ;

#endif
```

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    14/Jul/92  16:27:39

        Target option =  80C196
        Memory model  =  large
        Source file   =  io.c
        List file     =  io.lst
        Object file   =  io.rl5
        ASM file      =  io.sl5
        Command line  =  io.c -ml -vl -z -P -e -g -L -F -p52 -A
```

(c) Copyright Archimedes Software Inc. 1990

```
1   /*************************************************************************/
2   /*                                                                       */
3   /*   Module:    io.c                                                    */
4   /*                                                                       */
5   /*   Purpose:   The module contains procedures to perform low level I/O */
6   /*              (A/D, D/A, set vernier & gain, shifts).                  */
7   /*                                                                       */
8   /*   Inputs:    None.                                                   */
9   /*                                                                       */
10  /*   Outputs:   None.                                                   */
11  /*                                                                       */
12  /*   Uses:      80C196 specific definitions from including <io80C196.h>. */
13  /*                                                                       */
14  /*   History:   06_Feb_92   Written.                                    */
15  /*                                                                       */
16  /*************************************************************************/
17
18  #define  IO_C TRUE
19
20  #include <stdio.h>
21  #include <io80C196.h>
22  #include "types.h"
23  #include "debug.h"
24  #include "io.h"
25  #include "calib.h"
26  #include "modes.h"
27
28
29  void init_oces  ()
30  {                                   /* Select both instruments          */
31
32     SEL_BOTH () ;                    /* Initialize programmable timer    */
33
34     shift_oce_vals () ;              /* Set Preamp to range 4 (lowest gain) */
35
```

```
36                                          /* Set Vernier to 0x0100 (gain = 1.0)   */
37              SEL_EM () ;
38              EMeter_WR   = 0x0EFF ;
39              SFM_Vern_PG = 0x0EFF ;
40              ALP_Vern_PG = 0x0EFF ;
41                                          /* Shift values to Electrometer         */
42              shift_oce_vals () ;
43                                          /* Latch preamp and vernier gains       */
44              LATCH_PVG () ;
45                                          /* Unselect both instruments            */
46              UNSEL_BOTH() ;
47          }
```

```
48
49
50
51          void shift_oce_vals ()
52          {
53          #if DEBUG
54          if ((DEBUG_LEVEL & DEBUG_IO))
55            printf ("shift_oce_vals\n\r") ;
56          #endif
57                                        /* Initialize Timer#0 to 140uS envelope */
58                                        /* 0000|0011|00110000                   */
59                                        /* Control word: Counter 0,LSB then MSB */
60             Timer_WR = 0x0330 ;        /*               Mode 0, 16-bit binary  */
61
62                                        /* 0000|0000|00000100                   */
63             Timer_WR = 0x0004 ;        /* Counter LSB                          */
64
65                                        /* 0000|0000|00000010                   */
66             Timer_WR = 0x0002 ;        /* Counter MSB                          */
67
68                                        /* Initialize Timer#2 to 16-pulse wide  */
69                                        /* 0000|0011|10110000                   */
70                                        /* Control word: Counter 2,LSB then MSB */
71             Timer_WR = 0x03B0 ;        /*               Mode 0, 16-bit binary  */
72
73                                        /* 0000|0010|00010100                   */
74             Timer_WR = 0x020F ;        /* Counter LSB                          */
75
76                                        /* 0000|0010|00000001                   */
77             Timer_WR = 0x0200 ;        /* Counter MSB                          */
78
79                                        /* Initialize Timer#1 to 200kHz clock   */
80                                        /* 0000|0011|01110110                   */
81                                        /* Control word: Counter 1,LSB then MSB */
82             Timer_WR = 0x0376 ;        /*               Mode 3, 16-bit binary  */
83
84                                        /* 0000|0001|00010100                   */
85             Timer_WR = 0x0114 ;        /* Counter LSB                          */
86
87                                        /* 0000|0001|00000000                   */
88             Timer_WR = 0x0100 ;        /* Counter MSB                          */
89
90                                        /* Delay to allow allow Timer #0 to     */
91                                        /*    count 16 pulses                   */
92             delay (1) ;
93                                        /* Stop Timer#2 with mode 0 command     */
94                                        /* 0000|0011|01110000                   */
95             Timer_WR = 0x0370 ;
96
97           }
```

```
 98
 99
100
101        void set_grid   (BYTE grid_no, REAL voltage)
102        {
103           WORD dac_val ;
104                                          /* Convert voltage to 2's complement     */
105                                          /*    & add control info to high nibble  */
106           switch (grid_no)
107           {
108              case 1:  dac_val = (WORD) (G1_GEN_GAIN * voltage + G1_GEN_OFFSET) ;
109                       dac_val |= 0x4000 ;
110                       break ;
111              case 2:  dac_val = (WORD) (G2_GEN_GAIN * voltage + G2_GEN_OFFSET) ;
112                       dac_val |= 0x5000 ;
113                       break ;
114              case 3:  dac_val = (WORD) (G3_GEN_GAIN * voltage + G3_GEN_OFFSET) ;
115                       dac_val |= 0x6000 ;
116                       break ;
117              default: dac_val = (WORD) (G4_GEN_GAIN * voltage + G4_GEN_OFFSET) ;
118                       dac_val |= 0x7000 ;
119                       break ;
120           }
121                                          /* Write word to QUAD DAC                */
122           Grid_Volt_WR = dac_val ;
123
124        #if DEBUG
125        if ((DEBUG_LEVEL & DEBUG_IO))
126        printf ("set_grid #%d = %7.3f (0x%x)\n\r", grid_no, voltage, dac_val) ;
127        #endif
128        }
```

```
129
130
131
132         void set_grid_dac (BYTE grid_no, WORD dac_in)
133         {
134            WORD dac_val ;
135                                          /* Convert voltage to 2's complement     */
136                                          /*    & add control info to high nibble  */
137            switch (grid_no)
138            {
139               case 1:  dac_val = dac_in | 0x4000 ;
140                        break ;
141               case 2:  dac_val = dac_in | 0x5000 ;
142                        break ;
143               case 3:  dac_val = dac_in | 0x6000 ;
144                        break ;
145               default: dac_val = dac_in | 0x7000 ;
146                        break ;
147            }
148                                          /* Write word to QUAD DAC                */
149            Grid_Volt_WR = dac_val ;
150
151         #if DEBUG
152         if ((DEBUG_LEVEL & DEBUG_IO))
153         printf ("set_grid_dac #%d = 0x%x\n\r", grid_no, dac_val) ;
154         #endif
155         }
```

```
156
157
158
159     void set_dac4   (WORD dac_in)
160     {
161        WORD dac_val ;
162                                        /* Convert to 12-bit value              */
163        dac_val = 0x0FFF & dac_in ;
164                                        /* Add control info to high nibble      */
165        dac_val |= 0x7000 ;
166                                        /* Write word to QUAD DAC               */
167        Grid_Volt_WR = dac_val ;
168
169     #if DEBUG
170     if (DEBUG_LEVEL & DEBUG_IO)
171     printf ("set_dac4 = 0x%x (0x%x)\n\r", dac_in, dac_val) ;
172     #endif
173     }
```

```
174
175
176
177         void set_va     (BYTE instr, REAL voltage)
178         {
179            WORD dac_val ;
180                                          /* Convert voltage to 2's complement     */
181                                          /* Range: -65.534V to 65.536V            */
182            switch (instr)
183            {
184               case  SFM_INSTR :
185                        dac_val = (WORD) (SFM_VA_GAIN * voltage + SFM_VA_OFFSET) ;
186                        SFM_VA_Val = dac_val ;
187                        SEL_SFM () ;
188                        break ;
189               case  ALP_INSTR :
190                        dac_val = (WORD) (ALP_VA_GAIN * voltage + ALP_VA_OFFSET) ;
191                        ALP_VA_Val = dac_val ;
192                        SEL_ALP () ;
193                        break ;
194               case BOTH_INSTR :
195                        set_va (ALP_INSTR, voltage) ;
196                        set_va (SFM_INSTR, voltage) ;
197                        break ;
198               default: dac_val = 0x8000 ; /* Approximately 0 Volts                */
199                        SEL_BOTH () ;
200                        break ;
201            }
202                                          /* Select VA Generator                   */
203            SEL_VA () ;
204                                          /* Write word to 16-bit DAC              */
205            VAGen_WR = dac_val ;
206                                          /* Unselect both instruments             */
207            UNSEL_BOTH() ;
208
209         #if DEBUG
210         if ((DEBUG_LEVEL & DEBUG_IO))
211         printf ("set_va   #%d = %7.3f (0x%x)\n\r", instr, voltage, dac_val) ;
212         #endif
213         }
```

```
214
215
216
217         void set_va_dac (BYTE instr, WORD dac_val)
218         {
219                                     /* Make sure instrument is selected      */
220            UNSEL_BOTH () ;
221            IO_PORT1 |= instr ;
222                                     /* Select VA Generator                   */
223            SEL_VA () ;
224                                     /* Write word to 16-bit DAC              */
225            VAGen_WR = dac_val ;
226                                     /* Unselect both instruments             */
227            UNSEL_BOTH() ;
228
229            switch (instr)
230            {
231               case  SFM_INSTR : SFM_VA_Val = dac_val ;
232                                 break ;
233               case  ALP_INSTR : ALP_VA_Val = dac_val ;
234                                 break ;
235            }
236         #if DEBUG
237         if ((DEBUG_LEVEL & DEBUG_IO))
238         printf ("set_va_dac #%d = (0x%x)\n\r", instr, dac_val) ;
239         #endif
240         }
```

```
241
242
243
244         void set_vern_pg    (BYTE instr  , WORD value, BYTE val_type)
245         {
246            WORD vern_pg ;
247
248            if ((instr & SFM_INSTR) > 0)
249               vern_pg = SFM_Vern_PG ;
250            if ((instr & ALP_INSTR) > 0)
251               vern_pg = ALP_Vern_PG ;
252
253            switch (val_type)
254            {
255               case SET_VERN    : vern_pg &= 0xF000 ;
256                                  value    = ~(value) & 0x0FFF ;
257                                  vern_pg |= value  ;
258                                  break ;
259               case SET_PG      : vern_pg &= 0x0FFF ;
260                                  value    = value & 0xF000 ;
261                                  vern_pg |= value  ;
262                                  break ;
263               case SET_VERN_PG :
264               default          : vern_pg  = value  ;
265                                  break ;
266            }
267                                          /* Make sure instrument is selected      */
268            UNSEL_BOTH () ;
269            IO_PORT1 |= instr ;
270                                          /* Select Electrometer shift register    */
271            SEL_EM () ;
272                                          /* Write & latch gain values to OCE      */
273            EMeter_WR = vern_pg ;
274            shift_oce_vals () ;
275            LATCH_PVG () ;
276
277            if ((instr & SFM_INSTR) > 0)
278               SFM_Vern_PG = vern_pg ;
279            if ((instr & ALP_INSTR) > 0)
280               ALP_Vern_PG = vern_pg ;
281
282         #if DEBUG
283         if ((DEBUG_LEVEL & DEBUG_IO))
284         printf ("set_vern_pg 0x%X\n\r", vern_pg) ;
285         #endif
286         }
```

```
287
288
289
290     void clean_prb (BYTE instr, BYTE step     )
291     {
292        switch (step)
293        {
294           case 0:  if ((instr & SFM_INSTR) > 0)
295                    {
296                       SFM_CLEAN_ON () ;
297                       clean_word += SFM_CLN_1 ;
298                    }
299                    if ((instr & ALP_INSTR) > 0)
300                    {
301                       ALP_CLEAN_ON () ;
302                       clean_word += ALP_CLN_1 ;
303                    }
304                    break ;
305           case 1:  if ((instr & SFM_INSTR) > 0)
306                    {
307                       clean_word -= SFM_CLN_1 ;
308                       clean_word += SFM_CLN_2 ;
309                    }
310                    if ((instr & ALP_INSTR) > 0)
311                    {
312                       clean_word -= ALP_CLN_1 ;
313                       clean_word += ALP_CLN_2 ;
314                    }
315                    break ;
316           case 2:  if ((instr & SFM_INSTR) > 0)
317                       clean_word -= SFM_CLN_2 ;
318                    if ((instr & ALP_INSTR) > 0)
319                       clean_word -= ALP_CLN_2 ;
320                    break ;
321           default: printf ("\07Invalid step in clean_prb (%d)\n\r",step) ;
322                    break ;
323        }
324                                    /* Ensure high nibble is OK for decode   */
325           clean_word |= 0x8000 ;
326                                    /* Write cleaning word to latch          */
327           Probe_Cln_WR = clean_word ;
328        #if DEBUG
329        if ((DEBUG_LEVEL & DEBUG_IO))
330        printf ("clean_prb #%d step#%d (0x%x)\n\r", instr, step, clean_word) ;
331        #endif
332        }
```

```
333
334
335
336         void uncln_prb  (BYTE instr, BYTE step     )
337         {
338            switch (step)
339            {
340               case 0:  if ((instr & SFM_INSTR) > 0)
341                           clean_word += SFM_PRB_2 ;
342                        if ((instr & ALP_INSTR) > 0)
343                           clean_word += ALP_PRB_2 ;
344                        break ;
345               case 1:  if ((instr & SFM_INSTR) > 0)
346                        {
347                           clean_word -= SFM_PRB_2 ;
348                           clean_word += SFM_PRB_1 ;
349                        }
350                        if ((instr & ALP_INSTR) > 0)
351                        {
352                           clean_word -= ALP_PRB_2 ;
353                           clean_word += ALP_PRB_1 ;
354                        }
355                        break ;
356               case 2:  if ((instr & SFM_INSTR) > 0)
357                        {
358                           SFM_CLEAN_OFF () ;
359                           clean_word -= SFM_PRB_1 ;
360                        }
361                        if ((instr & ALP_INSTR) > 0)
362                        {
363                           ALP_CLEAN_OFF () ;
364                           clean_word -= ALP_PRB_1 ;
365                        }
366                        break ;
367               default: printf ("\07Invalid step in unclean_prb(%d)\n\r",step) ;
368                        break ;
369            }
370                                          /* Ensure high nibble is OK for decode   */
371            clean_word |= 0x8000 ;
372                                          /* Write cleaning word to latch          */
373            Probe_Cln_WR = clean_word ;
374
375         #if DEBUG
376         if ((DEBUG_LEVEL & DEBUG_IO))
377         printf ("uncln_prb #%d step #%d (0x%x)\n\r", instr, step, clean_word) ;
378         #endif
379         }
```

```
380
381
382
383         WORD read_ad    (BYTE instr                )
384         {
385
386         #if DEBUG
387         if ((DEBUG_LEVEL & DEBUG_IO))
388         printf ("read_ad #%d\n\r", instr) ;
389         #endif
390
391            instr = instr ;
392            return (0) ;
393         }
```

Errors: none
Warnings: none
Code size: 1467

```
364
365
366
367         void alp_hr_ne ()
368         {
369            if (ALP_Mode_Step == 0)
370            {
371               set_va (ALP_INSTR, 5.0) ;
372            }
373
374            if (ALP_Mode_Step < NUM_PVG_SET_STEPS)
375            {                              /* Set preamp & vernier gain near FS     */
376               set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_50_FS) ;
377            }
378         }
```

```
379
380
381
382         void alp_clean  ()
383         {
384            if (ALP_Mode_Step == 0)
385            {
386               set_va   (ALP_INSTR,   0.0) ;
387            }
                                          /* Perform switching to clean probe      */
388
389            if (ALP_Mode_Step <= 2)
390               clean_prb (ALP_INSTR, ALP_Mode_Step) ;
                                          /* Perform switching to do measurements  */
391
392            if (ALP_Mode_Step >= CLEANING_TIME+2)
393               uncln_prb (ALP_INSTR, ALP_Mode_Step - (CLEANING_TIME+2)) ;
394
395            if (ALP_Mode_Step >= (modelist[Curr_ALP_Mode].num_steps - 1))
396            {
397               ALP_packet_ready = TRUE ;
398            }
399         }
```

```
rrors: none
arnings: none
Code size: 2782
```

TM

```
/*************************************************************************/
/*                                                                       */
/*   Module:     tm.h                                                    */
/*                                                                       */
/*   Purpose:    The purpose of this header file is to define structures */
/*               that will be used for the PCM telemetry packets based on */
/*               the CCSDS standard specified in the BLUE BOOK.          */
/*                                                                       */
/*   Inputs:     None.                                                   */
/*                                                                       */
/*   Outputs:    None.                                                   */
/*                                                                       */
/*   Uses:       Definitions from "ccsds.h".                             */
/*                                                                       */
/*   History:    14_Jan_92   Written.                                    */
/*                                                                       */
/*************************************************************************/

#include "types.h"
#include "ccsds.h"

#define COL_SIZE       32          /* Size of Telemetry Column          */

#define PACKET_VER     0x0000      /* Telemetry Source Packet Format    */

#define PACKET_TYPE    0x0000      /* Telemetry Packet                  */

#define SECOND_HDR     0x0800      /* Secondary Header is PRESENT       */

#define SFM_PROC_ID     0x01F0     /* SFM Application Process ID        */
#define ALP_PROC_ID     0x020F     /* ALP Application Process ID        */
#define IDLE_PROC_ID    0x07FF     /* IDLE Process ID                   */
#define HSKP_PROC_ID    0x04A5.    /* Houskeeping Process ID            */

#define SFM_TM_LEN        704      /* Length of data buffer in SFM packet  */
#define ALP_TM_LEN        704      /* Length of data buffer in ALP packet  */
#define IDLE_TM_LEN        10      /* Length of data buffer in IDLE packet */
#define HSKP_TM_LEN        32      /* Length of data buffer in HSKP packet */

                                   /* Segmentation flags:               */
#define FIRST_SEG       0x4000     /*   01 = First Segment              */
#define CONT_SEG        0x0000     /*   00 = Continuation Segment       */
#define LAST_SEG        0x8000     /*   10 = Last Segment               */
#define UNSEGMENTED     0xC000     /*   11 = Unsegmented Packet         */


                                   /* Structure for Optional Secondary  */
                                   /*    header (per CCSDS standard)    */
typedef struct in_hdr_struct {
/* WORD version : 2    ; */        /* Version number of secondary header */
                                   /*   0 - only supported version number */
/* WORD hdr_len : 6    ; */        /* Total number of octets in header  */

   BYTE ccsds_reqrd    ;

   BYTE mode           ;           /* Mode number associated with data  */

   DBLWORD start_time  ;           /* Start time of measurement mode    */
```

```
} INSTR_HDR ;

define SCND_HDR_VER    0x00
define SCND_HDR_LEN    (sizeof(INSTR_HDR)-1)

typedef struct sfm_tm_struct {
   DBLWORD     sync_marker ;       /* Packet syncronization marker          */
   WORD        packet_id   ;       /* Packet Identification                 */
   WORD        pkt_seq_ctl ;       /* Packet Sequence Control               */
   WORD        packet_len  ;       /* Length of the packet                  */
   INSTR_HDR   instr_hdr   ;       /* Optional Secondary Header             */
   BYTE        data [SFM_TM_LEN    /* Buffer space for data                 */
                     + 2 ] ; /*    + one word for Error Detection         */
 } SFM_TM ;

typedef struct alp_tm_struct {
   DBLWORD     sync_marker ;       /* Packet syncronization marker          */
   WORD        packet_id   ;       /* Packet Identification                 */
   WORD        pkt_seq_ctl ;       /* Packet Sequence Control               */
   WORD        packet_len  ;       /* Length of the packet                  */
   INSTR_HDR   instr_hdr   ;       /* Optional Secondary Header             */
   BYTE        data [ALP_TM_LEN    /* Buffer space for data                 */
                     + 2 ] ; /*    + one word for Error Detection         */
 } ALP_TM ;

typedef struct hskp_tm_struct {
   DBLWORD     sync_marker ;       /* Packet syncronization marker          */
   WORD        packet_id   ;       /* Packet Identification                 */
   WORD        pkt_seq_ctl ;       /* Packet Sequence Control               */
   WORD        packet_len  ;       /* Length of the packet                  */
   INSTR_HDR   instr_hdr   ;       /* Optional Secondary Header             */
   BYTE        data [HSKP_TM_LEN]; /* Buffer space for data                 */
   WORD        crc_value   ;       /* Word for Packet Error Detection(CRC)  */
   HSKP_TM ;

typedef struct idle_tm_struct {
```

```
   DBLWORD     sync_marker ;       /* Packet syncronization marker        */

   WORD        packet_id   ;       /* Packet Identification               */

   WORD        pkt_seq_ctl ;       /* Packet Sequence Control             */

   WORD        packet_len  ;       /* Length of the packet                */

   INSTR_HDR   instr_hdr   ;       /* Optional Secondary Header           */

   WORD        crc_value   ;       /* Word for Packet Error Detection(CRC) */
}  IDLE_TM ;


extern ALP_TM    ALP_telem   ;
extern SFM_TM    SFM_telem   ;
extern HSKP_TM   HSKP_telem  ;
extern IDLE_TM   IDLE_telem  ;

#define IDLE_TM_TIME       250

#ifndef TM_C

extern BYTE ALP_packet_ready                    ;
extern BYTE SFM_packet_ready                    ;
extern BYTE HSKP_packet_ready                   ;
extern BYTE IDLE_packet_ready                   ;
extern BYTE send_serial_tm                      ;

extern WORD new_ALP_AD_val ;
extern WORD new_SFM_AD_val ;

extern WORD idle_count     ;

extern void data_to_tm      (                ) ;
extern void send_ready_tm   (                ) ;

extern void test_telem      (                ) ;
extern void test_telem_two_chan (            ) ;
extern void init_telem      (                ) ;

extern void send_telem      (BYTE *addr, WORD length) ;

#else

BYTE ALP_packet_ready                    ;
BYTE SFM_packet_ready                    ;
BYTE HSKP_packet_ready                   ;
BYTE IDLE_packet_ready                   ;
BYTE send_serial_tm                      ;

WORD new_ALP_AD_val ;
WORD new_SFM_AD_val ;

WORD idle_count     ;

void data_to_tm      (                ) ;
void send_ready_tm   (                ) ;

void test_telem_two_chan (            ) ;
```

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    14/Jul/92  16:25:30

        Target option =  80C196
        Memory model  =  large
        Source file   =  tm.c
        List file     =  tm.lst
        Object file   =  tm.rl5
        ASM file      =  tm.sl5
        Command line  =  tm.c -ml -vl -z -P -e -g -L -F -p52 -A
```

(c) Copyright Archimedes Software Inc. 1990

```
1          /*****************************************************************************/
2          /*                                                                           */
3          /*   Module:    tm.c                                                         */
4          /*                                                                           */
5          /*   Purpose:   This source file contains the routines used to test,         */
6          /*              and initialize the telemetry system and send assembled       */
7          /*              telemetry packets  specified in the BLUE BOOK.               */
8          /*                                                                           */
9          /*   Inputs:    None.                                                        */
10         /*                                                                           */
11         /*   Outputs:   None.                                                        */
12         /*                                                                           */
13         /*   Uses:      Definitions from "types.h", "ccsds.h", "tm.h".               */
14         /*                                                                           */
15         /*   History:   06_Jan_92   Written.                                         */
16         /*                                                                           */
17         /*****************************************************************************/
18
19         #define TM_C TRUE
20
21         #include <string.h>
22
23         #include <stdio.h>
24         #include <io80c196.h>
25         #include "types.h"
26         #include "debug.h"
27         #include "tm.h"
28         #include "modes_tm.h"
29         #include "io.h"
30
31
32
33         void data_to_tm      (                    )
34         {
35            SEL_BOTH () ;                    /* Select both instruments                */
```

```
36
37          SEL_EM () ;                    /* Select electrometers                    */
38
39          new_ALP_AD_val = EMeter_RD ;   /* Load AD values into shift registers    */
40
41          shift_oce_vals () ;            /* Shift values through optocouplers       */
42
43          UNSEL_SFM () ;
44                                         /* Read ALP A/D value                      */
45          new_ALP_AD_val = ~(EMeter_RD) & 0x0FFF ;
46
47          UNSEL_ALP () ;
48          SEL_SFM () ;
49                                         /* Read SFM A/D value                      */
50          new_SFM_AD_val = ~(EMeter_RD) & 0x0FFF ;
51
52        #if DEBUG
53        if ((DEBUG_LEVEL & DEBUG_MODES_TM))
54        printf ("SFM = 0x%x  ALP = 0x%x\n\r", new_SFM_AD_val, new_ALP_AD_val) ;
55        #endif
56
57           (*tm_fun_list [Curr_SFM_Mode].tm_fun) () ;
58
59           (*tm_fun_list [Curr_ALP_Mode].tm_fun) () ;
60
61        }
```

```
62
63
64
65          void send_ready_tm   (                    )
66          {
67             WORD p_len ;
68
69             if ( ALP_packet_ready)
70             {
71                p_len = 10 + ALP_telem.packet_len ;
72          #if DEBUG
73          if ((DEBUG_LEVEL & DEBUG_TM))
74          printf ("ALP  = %d\n\r", p_len) ;
75          #endif
76                send_telem ((BYTE *) &ALP_telem, p_len) ;
77                ALP_packet_ready = FALSE ;
78             }
79
80             if ( SFM_packet_ready)
81             {
82                p_len = 10 + SFM_telem.packet_len ;
83          #if DEBUG
84          if ((DEBUG_LEVEL & DEBUG_TM))
85          printf ("SFM  = %d\n\r", p_len, fill_len) ;
86          #endif
87                send_telem ((BYTE *) &SFM_telem, p_len) ;
88                SFM_packet_ready = FALSE ;
89             }
90
91             if (IDLE_packet_ready)
92             {
93                p_len = sizeof (IDLE_TM) ;
94          #if DEBUG
95          if ((DEBUG_LEVEL & DEBUG_TM))
96          printf ("IDLE  = %d\n\r", p_len) ;
97          #endif
98                send_telem ((BYTE *) &IDLE_telem, p_len) ;
99                IDLE_packet_ready = FALSE ;
100            }
101
102         /*
103            if (HSKP_packet_ready)
104            {
105               p_len = sizeof (HSKP_TM) ;
106         #if DEBUG
107         if ((DEBUG_LEVEL & DEBUG_TM))
108         printf ("HSKP = %d\n\r", p_len) ;
109         #endif
110               send_telem ((BYTE *) &HSKP_telem, p_len) ;
111               HSKP_packet_ready = FALSE ;
```

```
            }
        */
        }
```

```
115
116
117
118         void test_telem      (              )
119         {
120            WORD count ;
121            WORD count2 ;
122
123            for (count = 0 ; count <  800 ; count++)
124               Telem_WR = 0xFFFF ;
125            STR_WDOG () ;
126            delay (250) ;
127            for (count = 0 ; count <  800 ; count++)
128               Telem_WR = 0x0000 ;
129            STR_WDOG () ;
130            delay (250) ;
131            for (count = 0 ; count <  800 ; count++)
132               Telem_WR = 0x0100 ;
133            STR_WDOG () ;
134            delay (250) ;
135            for (count = 0 ; count < 512 ; count ++)
136            {
137               for (count2 = 0 ; count2 <512  ; count2++)
138                  Telem_WR = count ;
139               STR_WDOG () ;
140               delay (200) ;
141            }
142            for (count = 0 ; count < 512 ; count ++)
143               Telem_WR = count ;
144            STR_WDOG () ;
145            delay (200) ;
146            for (count = 0 ; count <  800 ; count++)
147               Telem_WR = 0x0000 ;
148            STR_WDOG () ;
149            delay (250) ;
150            for (count = 0 ; count <  800 ; count++)
151               Telem_WR = 0x0100 ;
152            STR_WDOG () ;
153            delay (250) ;
154            for (count = 0 ; count <  800 ; count++)
155               Telem_WR = 0xFFFF ;
156            STR_WDOG () ;
157
158            SEL_BOTH () ;
159            SEL_VA () ;
160
161            set_grid_dac (1, 0xFFF) ;
162            set_grid_dac (2, 0xFFF) ;
163            set_grid_dac (3, 0xFFF) ;
164            set_grid_dac (4, 0xFFF) ;
```

```
165             set_va_dac   (BOTH_INSTR, 0xFFFF) ;
166             delay (750) ; STR_WDOG () ;
167
168             set_grid_dac (1, 0x7FF) ;
169             set_grid_dac (2, 0x7FF) ;
170             set_grid_dac (3, 0x7FF) ;
171             set_grid_dac (4, 0x7FF) ;
172             set_va_dac   (BOTH_INSTR, 0x7FFF) ;
173             delay (750) ; STR_WDOG () ;
174
175             set_grid_dac (1, 0x000) ;
176             set_grid_dac (2, 0x000) ;
177             set_grid_dac (3, 0x000) ;
178             set_grid_dac (4, 0x000) ;
179             set_va_dac   (BOTH_INSTR, 0x0000) ;
180             delay (750) ; STR_WDOG () ;
181
182             set_grid (1         , 0.0) ;
183             set_grid (2         , 0.0) ;
184             set_grid (3         , 0.0) ;
185             set_dac4 (          0x800) ;
186             set_va   (BOTH_INSTR, 0.0) ;
187             STR_WDOG () ;
188
189         }
```

```
190
191
192
193         void test_telem_two_chan (          )
194         {
195            WORD count ;
196            WORD count2 ;
197
198
199            for (count = 0 ; count <  400 ; count++)
200            {
201               Telem_WR = 0xFFFF ;
202               Telem_WR = 0x0000 ;
203            }
204            STR_WDOG () ;
205            delay (250) ;
206            for (count = 0 ; count <  400 ; count++)
207            {
208               Telem_WR = 0x0000 ;
209               Telem_WR = 0xFFFF ;
210            }
211            STR_WDOG () ;
212            delay (250) ;
213            for (count = 0 ; count <  800 ; count++)
214               Telem_WR = 0x0100 ;
215            STR_WDOG () ;
216            delay (250) ;
217            for (count = 0 ; count < 512 ; count ++)
218            {
219               for (count2 = 0 ; count2 < 256  ; count2++)
220               {
221                  Telem_WR = count ;
222                  Telem_WR = 512 - count ;
223               }
224               STR_WDOG () ;
225               delay (200) ;
226            }
227            for (count = 0 ; count <  400 ; count++)
228            {
229               Telem_WR = 0x0000 ;
230               Telem_WR = 0xFFFF ;
231            }
232            STR_WDOG () ;
233            delay (250) ;
234            for (count = 0 ; count <  800 ; count++)
235               Telem_WR = 0x0100 ;
236            STR_WDOG () ;
237            delay (250) ;
238            for (count = 0 ; count <  400 ; count++)
239            {
```

```
240                 Telem_WR = 0xFFFF ;
241                 Telem_WR = 0x0000 ;
242              }
243              STR_WDOG () ;
244              delay (250) ;
245           }
```

```
246
247
248
249         void init_telem      (             )
250         {
251            WORD count ;
252
253                                          /* Do not send serial telemetry          */
254            send_serial_tm = FALSE ;
255                                          /* Initialize idle counter               */
256            idle_count     = 0     ;
257
258                                          /* Init frame sync marker                */
259            ALP_telem.sync_marker = CCSDS_SYNC_MARKER ;
260                                          /* Initialize Primary Header             */
261                                          /*    Initialize Packet Identification   */
262            ALP_telem.packet_id = PACKET_VER | PACKET_TYPE | SECOND_HDR
263                                            | ALP_PROC_ID ;
264                                          /*    Initialize Packet Sequence Control */
265                                          /*       Unsegmented Packet              */
266            ALP_telem.pkt_seq_ctl = UNSEGMENTED ;
267                                          /*    Initialize Packet Length           */
268            ALP_telem.packet_len = 0 ;
269                                          /* Initialize Secondary Header           */
270            ALP_telem.instr_hdr.ccsds_reqrd = SCND_HDR_VER | SCND_HDR_LEN ;
271            ALP_telem.instr_hdr.mode        = 0 ;
272            ALP_telem.instr_hdr.start_time  = 0 ;
273                                          /* Zero out data portion of packet       */
274            for (count = 0 ; count < ALP_TM_LEN+2 ; count++)
275               ALP_telem.data[count] = 0x00 ;
276            STR_WDOG () ;
277                                          /* Init frame sync marker                */
278            SFM_telem.sync_marker = CCSDS_SYNC_MARKER ;
279                                          /* Initialize Primary Header             */
280                                          /*    Initialize Packet Identification   */
281            SFM_telem.packet_id = PACKET_VER | PACKET_TYPE | SECOND_HDR
282                                            | SFM_PROC_ID ;
283                                          /*    Initialize Packet Sequence Control */
284                                          /*       Unsegmented Packet              */
285            SFM_telem.pkt_seq_ctl = UNSEGMENTED ;
286                                          /*    Initialize Packet Length           */
287            SFM_telem.packet_len = 0 ;
288                                          /* Initialize Secondary Header           */
289            SFM_telem.instr_hdr.ccsds_reqrd = SCND_HDR_VER | SCND_HDR_LEN ;
290            SFM_telem.instr_hdr.mode        = 0 ;
291            SFM_telem.instr_hdr.start_time  = 0 ;
292                                          /* Zero out data portion of packet       */
293            for (count = 0 ; count < SFM_TM_LEN+2 ; count++)
294               SFM_telem.data[count] = 0x00 ;
295            STR_WDOG () ;
```

```
296                                     /* Init frame sync marker                */
297         HSKP_telem.sync_marker = CCSDS_SYNC_MARKER ;
298                                     /* Initialize Primary Header             */
299                                     /*    Initialize Packet Identification   */
300         HSKP_telem.packet_id = PACKET_VER | PACKET_TYPE | SECOND_HDR
301                                       | HSKP_PROC_ID ;
302                                     /*    Initialize Packet Sequence Control */
303                                     /*       Unsegmented Packet              */
304         HSKP_telem.pkt_seq_ctl = UNSEGMENTED ;
305                                     /*    Initialize Packet Length           */
306         HSKP_telem.packet_len = 0 ;
307                                     /* Initialize Secondary Header           */
308         HSKP_telem.instr_hdr.ccsds_reqrd = SCND_HDR_VER | SCND_HDR_LEN ;
309         HSKP_telem.instr_hdr.mode        = 0 ;
310         HSKP_telem.instr_hdr.start_time  = 0 ;
311                                     /* Zero out data portion of packet       */
312         for (count = 0 ; count < HSKP_TM_LEN ; count++)
313            HSKP_telem.data[count] = 0x00 ;
314         HSKP_telem.crc_value = 0x0000 ;
315         STR_WDOG () ;
316
317                                     /* Init frame sync marker                */
318         IDLE_telem.sync_marker = CCSDS_SYNC_MARKER ;
319                                     /* Initialize Primary Header             */
320                                     /*    Initialize Packet Identification   */
321         IDLE_telem.packet_id = PACKET_VER | PACKET_TYPE | SECOND_HDR
322                                       | IDLE_PROC_ID ;
323                                     /*    Initialize Packet Sequence Control */
324                                     /*       Unsegmented Packet              */
325         IDLE_telem.pkt_seq_ctl = UNSEGMENTED ;
326                                     /*    Initialize Packet Length           */
327         IDLE_telem.packet_len = sizeof (         INSTR_HDR)  + 2 ;
328                                     /* Initialize Secondary Header           */
329         IDLE_telem.instr_hdr.ccsds_reqrd = SCND_HDR_VER | SCND_HDR_LEN ;
330         IDLE_telem.instr_hdr.mode        = 0xFF ;
331         IDLE_telem.instr_hdr.start_time  = 0 ;
332
333         IDLE_telem.crc_value = 0xFFFF ;
334
335         ALP_packet_ready  = FALSE ;
336         SFM_packet_ready  = FALSE ;
337         HSKP_packet_ready = FALSE ;
338         IDLE_packet_ready = FALSE ;
339      }
```

```
340
341
342
343         void send_telem      (BYTE *addr, WORD length)
344         {
345            if (send_serial_tm)
346            {
347               printf ("\n\r\07:") ;
348               while ((length--) > 0)
349                  printf ("%02x",*(addr++)) ;
350               printf ("\n\r") ;
351            }
352            else
353            {
354               while ((length--) > 0)
355                  Telem_WR = 0x0100 + *(addr++) ;
356            }
357         }
```

Errors: none
Warnings: none
Code size: 1909

```
/*****************************************************************************/
/*                                                                           */
/*   Module:     modes_tm.h                                                  */
/*                                                                           */
/*   Purpose:    The purpose of this header file is to define the struc-     */
/*               tures that will be used for telemetry from various modes.   */
/*                                                                           */
/*   Inputs:     None.                                                       */
/*                                                                           */
/*   Outputs:    None.                                                       */
/*                                                                           */
/*   Uses:       Type declarations from "types.h" and "modes.h"              */
/*                                                                           */
/*   History:    08_Feb_92   Written.                                        */
/*                                                                           */
/*****************************************************************************/

#include "types.h"
#include "modes.h"

typedef struct modes_tm_struct
{
   void (*tm_fun) () ;              /* Function call to stuff packet          */
} MODES_TM ;


#ifndef MODES_TM_C

extern void tm_sfm_std     () ;
extern void tm_sfm_hi_res  () ;
extern void tm_sfm_gt_1    () ;
extern void tm_sfm_gt_2    () ;
extern void tm_sfm_gt_3    () ;
extern void tm_sfm_irpa    () ;
extern void tm_sfm_erpa    () ;
extern void tm_sfm_clean   () ;
extern void tm_alp_5_pt    () ;
extern void tm_alp_std     () ;
extern void tm_alp_hi_res  () ;
extern void tm_alp_m_res   () ;
extern void tm_alp_hr_ne   () ;
extern void tm_alp_clean   () ;
extern void tm_null_mode   () ;
extern void tm_hskp_mode   () ;

extern MODES_TM tm_fun_list [LAST_ENTRY] ;


#else

void tm_sfm_std     () ;
void tm_sfm_hi_res  () ;
void tm_sfm_gt_1    () ;
void tm_sfm_gt_2    () ;
void tm_sfm_gt_3    () ;
void tm_sfm_irpa    () ;
void tm_sfm_erpa    () ;
void tm_sfm_clean   () ;
void tm_alp_5_pt    () ;
```

```
void tm_alp_std    () ;
void tm_alp_hi_res () ;
void tm_alp_m_res  () ;
void tm_alp_clean  () ;
void tm_alp_hr_ne  () ;
void tm_null_mode  () ;
void tm_hskp_mode  () ;

const MODES_TM tm_fun_list  [LAST_ENTRY] = {
   {tm_sfm_std   },
   {tm_sfm_hi_res},
   {tm_sfm_gt_1  },
   {tm_sfm_gt_2  },
   {tm_sfm_gt_3  },
   {tm_sfm_irpa  },
   {tm_sfm_erpa  },
   {tm_sfm_clean },
   {tm_alp_5_pt  },
   {tm_alp_std   },
   {tm_alp_hi_res},
   {tm_alp_m_res },
   {tm_alp_hr_ne },
   {tm_alp_clean },
   {tm_null_mode },
   {tm_hskp_mode } } ;
#endif
```

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    14/Jul/92  16:27:47

        Target option =  80C196
        Memory model  =  large
        Source file   =  modes_tm.c
        List file     =  modes_tm.lst
        Object file   =  modes_tm.rl5
        ASM file      =  modes_tm.sl5
        Command line  =  modes_tm.c -ml -vi -z -P -e -g -L -F -p52 -A
```

(c) Copyright Archimedes Software Inc. 1990

```
1       /*************************************************************************/
2       /*                                                                       */
3       /*     Name:     modes_tm.c                                              */
4       /*                                                                       */
5       /*     Params:   None.                                                   */
6       /*                                                                       */
7       /*     Input:    None.                                                   */
8       /*                                                                       */
9       /*     Output:                                                           */
10      /*                                                                       */
11      /*                                                                       */
12      /*     Purpose:                                                          */
13      /*                                                                       */
14      /*                                                                       */
15      /*     Mods:     Apr_01_92   NRV   Authored.                             */
16      /*                                                                       */
17      /*************************************************************************/
18
19      #define MODES_TM_C   TRUE
20
21      #include "types.h"
22      #include "debug.h"
23      #include <stdio.h>
24      #include "tm.h"
25      #include "io.h"
26      #include "crc.h"
27      #include "modes_tm.h"
28
29
30      /*************************************************************************/
31
32
33
34      void tm_sfm_std    ()
35      {
```

```
36          struct TM_SFM_STD {                 /* Sent   once   per mode              */
37             WORD pre_vern_gn       ;
38             WORD ad_val [ 57]      ;
39             WORD crc_value         ; } ;
40
41             struct TM_SFM_STD *data ;        /* Pointer to packet data              */
42
43                                              /* Get pointer to data part of packet   */
44             data = (struct TM_SFM_STD *) &(SFM_telem.data) ;
45             if (SFM_Mode_Step == 0)
46             {                                /* Compute length of telemetry packet   */
47                SFM_telem.packet_len = sizeof (         INSTR_HDR) +
48                                       sizeof (struct  TM_SFM_STD) ;
49                                              /* Place correct values in secondary    */
50                                              /*  instrument header                   */
51                SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
52                SFM_telem.instr_hdr.start_time = Clock_ZP   ;
53                                              /* Initialize crc value                 */
54                data->crc_value   = 0x1111 ;
55             }
56             else
57             {
58                if (SFM_Mode_Step >= NUM_PVG_SET_STEPS)
59                   data->ad_val[SFM_Mode_Step - NUM_PVG_SET_STEPS] = new_SFM_AD_val ;
60             }
61
62             if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
63             {
64                data->pre_vern_gn = SFM_Vern_PG ;
65                SFM_packet_ready  = TRUE   ;
66             }
67          }
```

```
68
69
70
71          void tm_sfm_hi_res ()
72          {
73          struct TM_SFM_HI_RES {            /* Sent   once   per mode              */
74             WORD pre_vern_gn       ;
75             WORD ad_val      [225] ;
76             WORD crc_value         ; } ;
77
78             struct TM_SFM_HI_RES *data ;    /* Pointer to packet data              */
79
80                                             /* Get pointer to data part of packet   */
81             data = (struct TM_SFM_HI_RES *) &(SFM_telem.data) ;
82
83             if (SFM_Mode_Step == 0)
84             {                               /* Compute length of telemetry packet   */
85                SFM_telem.packet_len = sizeof (         INSTR_HDR) +
86                                       sizeof (struct  TM_SFM_HI_RES) ;
87                                             /* Place correct values in secondary    */
88                                             /*   instrument header                  */
89                SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
90                SFM_telem.instr_hdr.start_time = Clock_ZP   ;
91
92                                             /* Initialize crc value                 */
93                data->crc_value   = 0x2222 ;
94             }
95             else
96             {
97                if (SFM_Mode_Step >= NUM_PVG_SET_STEPS)
98                   data->ad_val[SFM_Mode_Step - NUM_PVG_SET_STEPS] = new_SFM_AD_val ;
99             }
100
101            if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
102            {
103               data->pre_vern_gn = SFM_Vern_PG ;
104               SFM_packet_ready  = TRUE   ;
105            }
106         }
```

```
107
108
109
110     void tm_sfm_gt_1 ()
111     {
112     struct TM_SFM_GR_TEST {            /* Sent   once   per mode              */
113        WORD pre_vern_gn [     5] ;
114        WORD ad_val      [5] [29] ;
115        WORD crc_value           ; } ;
116
117        struct TM_SFM_GR_TEST *data ;   /* Pointer to packet data              */
118        WORD sweep_step ;
119        WORD sweep_num  ;
120
121                                        /* Get pointer to data part of packet  */
122        data = (struct TM_SFM_GR_TEST *) &(SFM_telem.data) ;
123
124        if (SFM_Mode_Step == 0)
125        {                               /* Compute length of telemetry packet  */
126           SFM_telem.packet_len = sizeof (         INSTR_HDR) +
127                                  sizeof (struct  TM_SFM_GR_TEST) ;
128                                        /* Place correct values in secondary   */
129                                        /*   instrument header                 */
130           SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
131           SFM_telem.instr_hdr.start_time = Clock_ZP   ;
132
133                                        /* Initialize crc value                */
134           data->crc_value      = 0x3333 ;
135        }
136        else
137        {
138           sweep_step = SFM_Mode_Step % 45 ;
139           sweep_num  = SFM_Mode_Step / 45 ;
140           if (sweep_step == NUM_PVG_SET_STEPS+1)
141              data->pre_vern_gn[sweep_num]  = SFM_Vern_PG ;
142
143           if (sweep_step >= NUM_PVG_SET_STEPS)
144              data->ad_val[sweep_num][sweep_step] = new_SFM_AD_val ;
145        }
146
147        if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
148                 SFM_packet_ready     = TRUE   ;
149     }
```

```
150
151
152
153     void tm_sfm_gt_2 ()
154     {
155     struct TM_SFM_GR_TEST {              /* Sent   once   per mode                  */
156        WORD pre_vern_gn [      5] ;
157        WORD ad_val      [5] [29] ;
158        WORD crc_value            ; } ;
159
160        struct TM_SFM_GR_TEST *data ;   /* Pointer to packet data                   */
161        WORD sweep_step ;
162        WORD sweep_num  ;
163                                        /* Get pointer to data part of packet       */
164
165        data = (struct TM_SFM_GR_TEST *) &(SFM_telem.data) ;
166
167        if (SFM_Mode_Step == 0)
168                                        /* Compute length of telemetry packet       */
169        {
170           SFM_telem.packet_len = sizeof (          INSTR_HDR) +
171                                  sizeof (struct  TM_SFM_GR_TEST) ;
172                                        /* Place correct values in secondary        */
173                                        /*   instrument header                      */
174
175           SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
176           SFM_telem.instr_hdr.start_time = Clock_ZP   ;
177
178                                        /* Initialize crc value                     */
179
180           data->crc_value      = 0x4444 ;
181
182        }
183        else
184        {
185           sweep_step = SFM_Mode_Step % 45 ;
186           sweep_num  = SFM_Mode_Step / 45 ;
187           if (sweep_step == NUM_PVG_SET_STEPS+1)
188              data->pre_vern_gn[sweep_num]  = SFM_Vern_PG ;
189
190           if (sweep_step >= NUM_PVG_SET_STEPS)
191              data->ad_val[sweep_num][sweep_step] = new_SFM_AD_val ;
192        }
193
194        if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
195           SFM_packet_ready     = TRUE   ;
196     }
```

```
193
194
195
196     void tm_sfm_gt_3 ()
197     {
198     struct TM_SFM_GR_TEST {            /* Sent   once   per mode              */
199        WORD pre_vern_gn [     5] ;
200        WORD ad_val      [5] [29] ;
201        WORD crc_value           ; } ;
202
203        struct TM_SFM_GR_TEST *data ;   /* Pointer to packet data              */
204        WORD sweep_step ;
205        WORD sweep_num  ;
206
207                                        /* Get pointer to data part of packet  */
208        data = (struct TM_SFM_GR_TEST *) &(SFM_telem.data) ;
209
210        if (SFM_Mode_Step == 0)
211        {                               /* Compute length of telemetry packet  */
212           SFM_telem.packet_len = sizeof (         INSTR_HDR) +
213                                  sizeof (struct  TM_SFM_GR_TEST) ;
214                                        /* Place correct values in secondary   */
215                                        /*   instrument header                 */
216           SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
217           SFM_telem.instr_hdr.start_time = Clock_ZP   ;
218
219                                        /* Initialize crc value                */
220           data->crc_value      = 0x5555 ;
221        }
222        else
223        {
224           sweep_step = SFM_Mode_Step % 45 ;
225           sweep_num  = SFM_Mode_Step / 45 ;
226           if (sweep_step == NUM_PVG_SET_STEPS+1)
227              data->pre_vern_gn[sweep_num]  = SFM_Vern_PG ;
228
229           if (sweep_step >= NUM_PVG_SET_STEPS)
230              data->ad_val[sweep_num][sweep_step] = new_SFM_AD_val ;
231        }
232
233        if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
234           SFM_packet_ready     = TRUE   ;
235     }
```

```
236
237
238
239         void tm_sfm_irpa   ()
240         {
241         struct TM_SFM_IRPA {              /* Sent   once   per mode              */
242            WORD pre_vern_gn      ;
243            WORD ad_val      [51] ;
244            WORD crc_value        ; } ;
245
246            struct TM_SFM_IRPA *data ;     /* Pointer to packet data              */
247
248                                           /* Get pointer to data part of packet  */
249            data = (struct TM_SFM_IRPA *) &(SFM_telem.data) ;
250
251            if (SFM_Mode_Step == 0)
252            {                              /* Compute length of telemetry packet  */
253               SFM_telem.packet_len = sizeof (         INSTR_HDR) +
254                                      sizeof (struct  TM_SFM_IRPA) ;
255                                           /* Place correct values in secondary   */
256                                           /*   instrument header                 */
257               SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
258               SFM_telem.instr_hdr.start_time = Clock_ZP   ;
259
260                                           /* Initialize crc value                */
261               data->crc_value   = 0x6666 ;
262            }
263            else
264            {
265               if (SFM_Mode_Step >= NUM_PVG_SET_STEPS)
266                  data->ad_val[SFM_Mode_Step - NUM_PVG_SET_STEPS] = new_SFM_AD_val ;
267            }
268
269            if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
270            {
271               data->pre_vern_gn = SFM_Vern_PG ;
272               SFM_packet_ready  = TRUE   ;
273            }
274         }
```

```
275
276
277
278        void tm_sfm_erpa   ()
279        {
280        struct TM_SFM_ERPA {                /* Sent   once   per mode                  */
281           WORD pre_vern_gn      ;
282           WORD ad_val      [51] ;
283           WORD crc_value        ; } ;
284
285           struct TM_SFM_ERPA *data ;       /* Pointer to packet data                  */
286
287                                            /* Get pointer to data part of packet      */
288           data = (struct TM_SFM_ERPA *) &(SFM_telem.data) ;
289
290           if (SFM_Mode_Step == 0)
291                                            /* Compute length of telemetry packet      */
292           {
293              SFM_telem.packet_len = sizeof (         INSTR_HDR) +
294                                     sizeof (struct  TM_SFM_ERPA) ;
295                                            /* Place correct values in secondary       */
296                                            /*   instrument header                     */
297              SFM_telem.instr_hdr.mode       = modelist[Curr_SFM_Mode].mode_tm_id ;
298              SFM_telem.instr_hdr.start_time = Clock_ZP   ;
299
300                                            /* Initialize crc value                    */
301              data->crc_value   = 0x7777 ;
302           }
303           else
304           {
305              if (SFM_Mode_Step >= NUM_PVG_SET_STEPS)
306                 data->ad_val[SFM_Mode_Step - NUM_PVG_SET_STEPS] = new_SFM_AD_val ;
307           }
308
309           if (SFM_Mode_Step == (modelist[Curr_SFM_Mode].num_steps - 1))
310           {
311              data->pre_vern_gn = SFM_Vern_PG ;
312              SFM_packet_ready  = TRUE   ;
313           }
        }
```

```
314
315
316
317         void tm_sfm_clean  ()
318         {
319         struct TM_SFM_CLEAN {                /* Sent   once   per mode                    */
320            WORD empty            ;
321            WORD crc_value        ; } ;
322
323            struct TM_SFM_CLEAN *data ;      /* Pointer to packet data                    */
324
325                                             /* Get pointer to data part of packet        */
326            data = (struct TM_SFM_CLEAN *) &(SFM_telem.data) ;
327
328            if (SFM_Mode_Step == 2)
329                                             /* Compute length of telemetry packet        */
330            {
331               SFM_telem.packet_len = sizeof (          INSTR_HDR) +
332                                      sizeof (struct  TM_SFM_CLEAN) ;
333                                             /* Place correct values in secondary         */
334                                             /*   instrument header                       */
335               SFM_telem.instr_hdr.mode        = modelist[Curr_SFM_Mode].mode_tm_id ;
336               SFM_telem.instr_hdr.start_time = Clock_ZP   ;
337
338                                             /* Initialize crc value                      */
339               data->crc_value   = 0x8888 ;
340               data->empty       = 0x8888 ;
341            }
342
343            if (SFM_Mode_Step == (modelist[Curr_SFM_Mode ].num_steps - 1))
344            {
345               SFM_packet_ready  = TRUE   ;
346            }
         }
```

```
347
348
349
350     void tm_alp_5_pt   ()
351     {                                   /* Sent   once   per mode                    */
352     struct TM_ALP_5_PT {
353        WORD pre_vern_gn ;
354        WORD v1_conv, i1_conv ;
355        WORD v2_conv, i2_conv ;
356        WORD v3_conv, i3_conv ;
357        WORD v4_conv, i4_conv ;
358        WORD v5_conv, i5_conv ;
359        WORD kTe              ;
360        WORD crc_value        ; } ;
361
362        struct TM_ALP_5_PT *data ;       /* Pointer to packet data                   */
363
364                                         /* Get pointer to data part of packet       */
365        data = (struct TM_ALP_5_PT *) &(ALP_telem.data) ;
366
367        if (ALP_Mode_Step == 0)
368                                         /* Compute length of telemetry packet       */
369        {
370           ALP_telem.packet_len = sizeof (          INSTR_HDR) +
371                                   sizeof (struct  TM_ALP_5_PT) ;
372                                         /* Place correct values in secondary        */
373                                         /*   instrument header                      */
374           ALP_telem.instr_hdr.mode        = modelist[Curr_ALP_Mode].mode_tm_id ;
375           ALP_telem.instr_hdr.start_time = Clock_ZP   ;
376
377                                         /* Initialize crc value                     */
378           data->crc_value  = 0x9999 ;
379        }
380        else
381        {
382           switch (ALP_Mode_Step)
383           {
384              case 15 : data->pre_vern_gn = ALP_Vern_PG ;
385                        data->v1_conv = v1_conv ; data->i1_conv = i1_conv ;
386                        break ;
387              case 32 : data->v3_conv = v3_conv ; data->i3_conv = i3_conv ;
388                        break ;
389              case 48 : data->v4_conv = v4_conv ; data->i4_conv = i4_conv ;
390                        data->kTe     = kTe     ;
391                        break ;
392              case 49 : data->v2_conv = v2_conv ; data->i2_conv = i2_conv ;
393                        data->v5_conv = v5_conv ; data->i5_conv = i5_conv ;
394                        break ;
395           }
396        }
```

```
397         if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
398            ALP_packet_ready = TRUE ;
399      }
```

```
400
401
402
403     void tm_alp_std    ()
404     {                                  /* Sent   once   per mode                    */
405     struct TM_ALP_STD {
406        WORD pre_vern_gn        ;
407        WORD ad_val        [51] ;
408        WORD crc_value          ; } ;
409
410        struct TM_ALP_STD *data ;       /* Pointer to packet data                    */
411
412                                        /* Get pointer to data part of packet        */
413        data = (struct TM_ALP_STD *) &(ALP_telem.data) ;
414
415        if (ALP_Mode_Step == 0)          /* Compute length of telemetry packet        */
416        {
417           ALP_telem.packet_len = sizeof (         INSTR_HDR) +
418                                  sizeof (struct  TM_ALP_STD) ;
419                                        /* Place correct values in secondary         */
420                                        /*   instrument header                       */
421           ALP_telem.instr_hdr.mode        = modelist[Curr_ALP_Mode].mode_tm_id ;
422           ALP_telem.instr_hdr.start_time = Clock_ZP   ;
423
424                                        /* Initialize crc value                      */
425           data->crc_value  = 0xAAAA ;
426        }
427        else
428        {
429           if (ALP_Mode_Step >= NUM_PVG_SET_STEPS)
430              data->ad_val[ALP_Mode_Step - NUM_PVG_SET_STEPS] = new_ALP_AD_val ;
431        }
432
433        if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
434        {
435           data->pre_vern_gn = ALP_Vern_PG ;
436           ALP_packet_ready = TRUE   ;
437        }
438     }
```

```
439
440
441
442     void tm_alp_hi_res ()
443     {
444     struct TM_ALP_HI_RES {              /* Sent   once   per mode                    */
445        WORD pre_vern_gn       ;
446        WORD ad_val       [ 87] ;
447        WORD crc_value         ; } ;
448
449        struct TM_ALP_HI_RES *data ;    /* Pointer to packet data                    */
450
451                                        /* Get pointer to data part of packet        */
452        data = (struct TM_ALP_HI_RES *) &(ALP_telem.data) ;
453
454        if (ALP_Mode_Step == 0)
455                                        /* Compute length of telemetry packet        */
456        {
457           ALP_telem.packet_len = sizeof (         INSTR_HDR) +
458                                   sizeof (struct  TM_ALP_HI_RES) ;
459                                        /* Place correct values in secondary         */
460                                        /*   instrument header                       */
461
462           ALP_telem.instr_hdr.mode       = modelist[Curr_ALP_Mode].mode_tm_id ;
463           ALP_telem.instr_hdr.start_time = Clock_ZP   ;
464
465                                        /* Initialize crc value                      */
466
467           data->crc_value  = 0xBBBB ;
468        }
469        else
470        {
471           if (ALP_Mode_Step >= NUM_PVG_SET_STEPS)
472              data->ad_val[ALP_Mode_Step - NUM_PVG_SET_STEPS] = new_ALP_AD_val ;
473
474        }
475
476        if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
477        {
478           data->pre_vern_gn = ALP_Vern_PG ;
479           ALP_packet_ready = TRUE   ;
480        }
481     }
```

```
478
479
480
481     void tm_alp_m_res  ()
482     {
483     struct TM_ALP_M_RES {                /* Sent   once   per mode                    */
484        WORD pre_vern_gn       ;
485        WORD kTe               ;
486        WORD ad_val       [82] ;
487        WORD crc_value         ; } ;
488
489        struct TM_ALP_M_RES *data ;     /* Pointer to packet data                    */
490
491                                        /* Get pointer to data part of packet        */
492        data = (struct TM_ALP_M_RES *) &(ALP_telem.data) ;
493
494        if (ALP_Mode_Step == 0)
495                                        /* Compute length of telemetry packet        */
496        {
497           ALP_telem.packet_len = sizeof (         INSTR_HDR) +
498                                   sizeof (struct  TM_ALP_M_RES) ;
499                                        /* Place correct values in secondary         */
500                                        /*   instrument header                       */
501           ALP_telem.instr_hdr.mode       = modelist[Curr_ALP_Mode].mode_tm_id ;
502           ALP_telem.instr_hdr.start_time = Clock_ZP   ;
503
504                                        /* Initialize crc value                      */
505           data->crc_value  = 0xCCCC ;
506        }
507        else
508        {
509           if (ALP_Mode_Step >= NUM_PVG_SET_STEPS)
510              data->ad_val[ALP_Mode_Step - NUM_PVG_SET_STEPS] = new_ALP_AD_val ;
511        }
512
513        if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
514        {
515           data->pre_vern_gn = ALP_Vern_PG ;
516           data->kTe         = kTe         ;
517           ALP_packet_ready  = TRUE        ;
518        }
     }
```

```
519
520
521
522     void tm_alp_hr_ne  ()
523     {
524     struct TM_ALP_HR_NE {                /* Sent   once   per mode                    */
525        WORD pre_vern_gn       ;
526        WORD ad_val       [250] ;
527        WORD crc_value         ; } ;
528
529        struct TM_ALP_HR_NE *data ;      /* Pointer to packet data                    */
530
531                                         /* Get pointer to data part of packet        */
532        data = (struct TM_ALP_HR_NE *) &(ALP_telem.data) ;
533
534        if (ALP_Mode_Step == 0)
535                                         /* Compute length of telemetry packet        */
536        {
537           ALP_telem.packet_len = sizeof (          INSTR_HDR) +
538                                   sizeof (struct  TM_ALP_HR_NE) ;
539                                         /* Place correct values in secondary         */
540                                         /*   instrument header                       */
541           ALP_telem.instr_hdr.mode       = modelist[Curr_ALP_Mode].mode_tm_id ;
542           ALP_telem.instr_hdr.start_time = Clock_ZP   ;
543
544                                         /* Initialize crc value                      */
545           data->crc_value  = 0xDDDD ;
546        }
547        else
548        {
549           if (ALP_Mode_Step >= NUM_PVG_SET_STEPS)
550              data->ad_val[ALP_Mode_Step - NUM_PVG_SET_STEPS] = new_ALP_AD_val ;
551        }
552
553        if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
554        {
555           data->pre_vern_gn = ALP_Vern_PG ;
556           ALP_packet_ready = TRUE   ;
557        }
        }
```

```
558
559
560
561    void tm_alp_clean  ()
562    {
563    struct TM_ALP_CLEAN {                /* Sent   once   per mode                */
564       WORD empty            ;
565       WORD crc_value        ; } ;
566
567       struct TM_ALP_CLEAN *data ;       /* Pointer to packet data                */
568
569                                         /* Get pointer to data part of packet    */
570       data = (struct TM_ALP_CLEAN *) &(ALP_telem.data) ;
571
572       if (ALP_Mode_Step == 3)
573                                         /* Compute length of telemetry packet    */
574       {
575          ALP_telem.packet_len = sizeof (         INSTR_HDR) +
576                                 sizeof (struct  TM_ALP_CLEAN) ;
577                                         /* Place correct values in secondary     */
578                                         /*   instrument header                   */
579          ALP_telem.instr_hdr.mode        = modelist[Curr_ALP_Mode].mode_tm_id ;
580          ALP_telem.instr_hdr.start_time = Clock_ZP   ;
581
582                                         /* Initialize crc value                  */
583          data->crc_value  = 0xEEEE ;
584          data->empty      = 0xEEEE ;
585       }
586       if (ALP_Mode_Step == (modelist[Curr_ALP_Mode].num_steps - 1))
587       {
588          ALP_packet_ready = TRUE   ;
589       }
    }
```

```
590
591
592
593        void tm_null_mode ()
594        {                                  /* Sent once per mode                    */
595        struct TM_NULL_MODE {
596           WORD empty              ;
597           WORD crc_value          ; } ;
598
599        }
```

```
600
601
602
603        void tm_hskp_mode  ()
604        {
605        struct TM_HSKP_MODE {              /* Sent   once   per mode              */
606           WORD empty            ;
607           WORD crc_value        ; } ;
608
609        }
```

Errors: none
Warnings: none
Code size: 2786

CCSDS.H

```
/*************************************************************************/
/*                                                                       */
/*   Module:     ccsds.h                                                 */
/*                                                                       */
/*   Purpose:    The purpose of this header file is to define structures */
/*               needed to use the CCSDS packet telemetry standard.      */
/*                                                                       */
/*   Inputs:     None.                                                   */
/*                                                                       */
/*   Outputs:    None.                                                   */
/*                                                                       */
/*   Uses:       Nothing from other files.                               */
/*                                                                       */
/*   History:    14_Jan_92   Written.                                    */
/*                                                                       */
/*************************************************************************/

#include "types.h"


/*************************************************************************/
/*                                                                       */
/*                    Source Packet Format -- Version 1                  */
/*                                                                       */
/*   Refer to the BLUE BOOK (CCSDS 102.0-B-2 Jan 1987) pp.3-1 to 4-6     */
/*                                                                       */
/*                                                                       */
/*   Note: Conventions used are:  bit 0 = MSB = first bit transmitted.   */
/*         Fields are grouped in 8-bit "words" or OCTETS.  Throughout    */
/*         this header file a WORD refers to 16-bit fields.              */
/*                                                                       */
/*************************************************************************/

                                /* 32-bit syncronization marker used to */
                                /*   immediately precede the Transfer   */
                                /*   Frame Header when Reed-Solomon     */
                                /*   IS NOT USED                        */
/* #define CCSDS_SYNC_MARKER  0x1ACFFC1D */
/* Bytes were reversed to correct for LSB first storage pattern of 80c196  */

#define CCSDS_SYNC_MARKER 0x1DFCCF1A


typedef struct PACKET_ID_STRUCT {
   WORD version     :  3 ;        /* Version # of the Source Packet:      */
                                  /*    0 = Telemetry Source Packet       */
                                  /*    4 = Telemetry Segment Format      */
   WORD type        :  1 ;        /* Type of Packet                       */
                                  /*    0 = Telemetry Packet              */
                                  /*    1 = Telecommand Packet            */
   WORD sec_hdr_flg :  1 ;        /* Secondary Header Flag                */
                                  /*    0 = Absent                        */
                                  /*    1 = Present                       */
   WORD app_proc_id : 11 ;        /* Application process ID: Uniquely     */
                                  /*   identifies process/instrument on   */
                                  /*   vehicle that created source packet */
                                  /*   All 1's reserved for IDLE PACKETS  */
 } PACKET_ID ;
```

```
typedef struct PKT_SEQ_CTL_STRUCT {
   WORD seq_flags   :  2 ;          /* Segmentation flags:                  */
                                    /*   00 = Continuation Segment          */
                                    /*   01 = First Segment                 */
                                    /*   10 = Last Segment                  */
                                    /*   11 = Unsegmented Packet            */
   WORD pkt_seq_cnt : 14 ;          /* Straight sequential count (mod 16384) */
                                    /*   of packet generated by each unique */
                                    /*   source application process         */
} PKT_SEQ_CTL ;
```

```
/*************************************************************************/
/*                                                                       */
/*   Module:    crc.h                                                    */
/*                                                                       */
/*   Purpose:   This file include a library of routines used to compute  */
/*              the CRC (cyclic redundancy check) code for a byte,       */
/*              or a buffer (array of bytes).  This error detection      */
/*              encoding scheme corresponds the the HDLC(ISO), ADCCP(ANSI) */
/*              V.41 (CCITT) and the 102.0-B-2 (CCSDS) standards.        */
/*                                                                       */
/*   History:   26_Jan_91   NRV   Written.                               */
/*                                                                       */
/*************************************************************************/


WORD crc_byte (BYTE byte_in, WORD crc) ;
WORD crc_word (WORD word_in, WORD crc) ;
WORD crc_buff (BYTE *buff_in, WORD buff_len, WORD crc) ;
```

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    13/Jul/92  04:03:21

        Target option =  80C196
        Memory model  =  large
        Source file   =  crc.c
        List file     =  crc.lst
        Object file   =  crc.rl5
        ASM file      =  crc.sl5
        Command line  =  crc.c -ml -vl -z -P -e -g -L -F -p52 -A
```

(c) Copyright Archimedes Software Inc. 1990

```
1   /*************************************************************************/
2   /*                                                                       */
3   /*   Module:    crc.c                                                   */
4   /*                                                                       */
5   /*   Purpose:   This file include a library of routines used to compute */
6   /*              the CRC (cyclic redundancy check) code for a byte,       */
7   /*              or a buffer (array of bytes).  This error detection      */
8   /*              encoding scheme corresponds the the HDLC(ISO), ADCCP(ANSI) */
9   /*              V.41 (CCITT) and the 102.0-B-2 (CCSDS) standards.        */
10  /*                                                                       */
11  /*   History:   26_Jan_91   NRV   Written.                               */
12  /*                                                                       */
13  /*************************************************************************/
14
15  #include "types.h"
16  #include "debug.h"
17
18  #define FEEDBACK 0x8408
19
20  #define ODD_BYTE(b)      ((b & 0x01  ) == 1)
21  #define ODD_WORD(w)      ((w & 0x0001) == 1)
22
23
24  WORD crc_byte (BYTE byte_in, WORD crc)
25  {
26     int c2 ;
27
28     for (c2 = 0 ; c2 < 8 ; c2++)
29     {
30        if (ODD_BYTE(byte_in) ^ ODD_WORD(crc))
31           crc = (crc >> 1) ^ FEEDBACK ;
32        else
33           crc >>= 1 ;
34        byte_in >>= 1 ;
35     }
```

```
36
37          return (crc) ;
38      }
```

```
39
40
41
42          WORD crc_word (WORD word_in, WORD crc)
43          {
44             WORD c2      ;
45             BYTE byte_in ;
46
47             byte_in = (BYTE) ((word_in & 0xFF00) >> 8) ;
48             for (c2 = 0 ; c2 < 8 ; c2++)
49             {
50                if (ODD_BYTE(byte_in) ^ ODD_WORD(crc))
51                   crc = (crc >> 1) ^ FEEDBACK ;
52                else
53                   crc >>= 1 ;
54                byte_in >>= 1 ;
55             }
56
57             byte_in = (BYTE) (word_in & 0x00FF) ;
58             for (c2 = 0 ; c2 < 8 ; c2++)
59             {
60                if (ODD_BYTE(byte_in) ^ ODD_WORD(crc))
61                   crc = (crc >> 1) ^ FEEDBACK ;
62                else
63                   crc >>= 1 ;
64                byte_in >>= 1 ;
65             }
66
67             return (crc) ;
68          }
```

```
69
70
71
72      WORD crc_buff (BYTE *buff_in, WORD buff_len, WORD crc)
73      {
74         int c1, c2 ;
75
76         for (c1 = 0 ; c1 < buff_len ; c1++)
77         {
78            for (c2 = 0 ; c2 < 8 ; c2++)
79            {
80               if (ODD_BYTE(buff_in[c1]) ^ ODD_WORD(crc))
81                  crc = (crc >> 1) ^ FEEDBACK ;
82               else
83                  crc >>= 1 ;
84               buff_in[c1] >>= 1 ;
85            }
86            buff_in[c1] >>= 1 ;
87         }
88
89         return (crc) ;
90      }
```

```
91

Errors: none
Warnings: none
Code size: 571
```

MODES

```
/*****************************************************************************/
/*                                                                           */
/*   Module:    modes.h                                                      */
/*                                                                           */
/*   Purpose:   The purpose of this header file is to define the mode       */
/*              definition tables.                                           */
/*                                                                           */
/*   Inputs:    None.                                                        */
/*                                                                           */
/*   Outputs:   None.                                                        */
/*                                                                           */
/*   Uses:      Nothing from other files.                                    */
/*                                                                           */
/*   History:   14_Feb_92   Written.                                         */
/*                                                                           */
/*****************************************************************************/

#include "types.h"

typedef struct mode_struct
{
   BYTE  mode_tm_id ;              /* Mode ID for telemetry                */
   char *lname      ;              /* Long name of mode (40 chars)         */
   char *sname      ;              /* Short abbreviated name for mode      */
   void (*mode_fun) () ;           /* Function call to execute mode        */
   WORD  num_steps  ;              /* Number of steps in mode              */
} MODE ;

                                   /* References to variables defined to   */
                                   /*    be in ZRAM in SFMALP.ASM          */
extern WORD ALP_Mode_Ptr ;
extern WORD SFM_Mode_Ptr ;

extern BYTE Curr_ALP_Mode ;
extern BYTE Curr_SFM_Mode ;

extern WORD ALP_Mode_Step ;
extern WORD SFM_Mode_Step ;

extern DBLWORD SFM_mode_start_time ;
extern DBLWORD ALP_mode_start_time ;

enum mode_names { SFM_STD, SFM_HR , SFM_GT1, SFM_GT2, SFM_GT3, SFM_IRP,
                  SFM_ERP, SFM_CLN,
                  ALP_5PT, ALP_STD, ALP_HR , ALP_MR , ALP_Ne , ALP_CLN,
                  NULL_MD, HSKP_MD, LAST_ENTRY } ;

#define TICKS_PER_SECOND   250

#define CLEANING_TIME  (5*TICKS_PER_SECOND) /* 5 second delay for cleaning */


#define NUM_PVG_SET_STEPS     16
#define NUM_VCONVERGE_STEPS   17

define NEAR_95_FS          0x0F99   /* 95.0% */
#define NEAR_50_FS          0x0C00   /* 50.0% */
#define NEAR_5PT_VAL        0x055D   /* -3.3V */

#ifndef MODES_C
```

```
extern MODE modelist [LAST_ENTRY] ;

 xtern void next_mode_step () ;
 xtern void init_modes     () ;

extern void set_mode_pvg (BYTE step, BYTE instr, WORD target_ad_val) ;

extern void vconverge (BYTE step, BYTE instr, WORD target_ad_val) ;

extern BYTE pg_converged  ;
extern WORD vern_conv_val ;
extern WORD vern_sar_step ;
extern WORD va_conv_val   ;
extern WORD va_sar_step   ;


Helse

void next_mode_step () ;
void init_modes     () ;

void sfm_std    () ;
void sfm_hi_res () ;
void sfm_gt_1   () ;
void sfm_gt_2   () ;
void sfm_gt_3   () ;
void sfm_irpa   () ;
void sfm_erpa   () ;
 )id sfm_clean  () ;
void alp_5_pt   () ;
void alp_std    () ;
void alp_hi_res () ;
void alp_m_res  () ;
void alp_hr_ne  () ;
void alp_clean  () ;

void null_mode  () ;
void hskp_mode  () ;

void set_mode_pvg (BYTE step, BYTE instr, WORD target_ad_val) ;

void vconverge (BYTE step, BYTE instr, WORD target_ad_val) ;

BYTE pg_converged  ;
  WORD vern_conv_val ;
WORD vern_sar_step ;
WORD va_conv_val   ;
WORD va_sar_step   ;

const MODE modelist [LAST_ENTRY] = {
{0x23,"Standard SFM Measurement Mode          ","SFM_STD", sfm_std   ,  73},
{0x2B,"High Resolution SFM Measurement Mode   ","SFM_HR ", sfm_hi_res, 241},
{0x26,"SFM Grid 1 Test Mode                   ","SFM_GT1", sfm_gt_1  , 226},
 0x29,"SFM Grid 2 Test Mode                   ","SFM_GT2", sfm_gt_2  , 226},
 0x25,"SFM Grid 3 Test Mode                   ","SFM_GT3", sfm_gt_3  , 226},
{0x87,"SFM Ion RPA Mode                       ","SFM_IRP", sfm_irpa  ,  67},
{0x8C,"SFM Electron RPA Modee                 ","SFM_ERP", sfm_erpa  ,  67},
{0x2E,"SFM Cleaning Mode                      ","SFM_CLN", sfm_clean ,5+CLEANING_TIME},
{0x40,"Five Point ALP Mode                    ","ALP_5PT", alp_5_pt  ,  51},
```

```
{0x41,"Standard ALP Sweep Mode                ","ALP_STD", alp_std   ,  67},
{0x42,"High Resolution ALP Sweep Mode         ","ALP_HR ", alp_hi_res, 103},
{0x44,"Medium Resolution ALP Sweep Mode       ","ALP_MR ", alp_m_res ,  97},
{0x4A,"High Resolution Ne Mode                ","ALP_Ne ", alp_hr_ne , 266},
{0x48,"ALP Cleaning Mode                      ","ALP_CLN", alp_clean ,5+CLEANING_TIME},
{0xFF,"NULL Mode                              ","NULL_MD", null_mode ,   4},
{0xF0,"Housekeeping Mode                      ","HSKP_MD", hskp_mode ,   1}};

#endif

#ifdef MODES2_C

WORD v1_conv = (WORD) (5.0 * ALP_VA_GAIN + ALP_VA_OFFSET) ;
WORD         i1_conv ;
REAL         i1_real ;
WORD v2_conv, i2_conv ;
WORD v3_conv, i3_conv ;
WORD v4_conv, i4_conv ;
WORD v5_conv, i5_conv ;

WORD kTe             ;

#else

extern WORD v1_conv, i1_conv ;
extern WORD v2_conv, i2_conv ;
extern WORD v3_conv, i3_conv ;
extern WORD v4_conv, i4_conv ;
extern WORD v5_conv, i5_conv ;

extern WORD kTe             ;

#endif
```

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    14/Jul/92  16:28:07

        Target option =  80C196
        Memory model  =  large
        Source file   =  modes2.c
        List file     =  modes2.lst
        Object file   =  modes2.rl5
        ASM file      =  modes2.sl5
        Command line  =  modes2.c -ml -vl -z -P -e -g -L -F -p52 -A

                                        (c) Copyright Archimedes Software Inc. 1990
```

```
1       /*************************************************************************/
2       /*                                                                       */
3       /*     Name:     modes.c                                                 */
4       /*                                                                       */
5       /*     Params:   None.                                                   */
6       /*                                                                       */
7       /*     Input:    None.                                                   */
8       /*                                                                       */
9       /*     Output:                                                           */
10      /*                                                                       */
11      /*                                                                       */
12      /*     Purpose:                                                          */
13      /*                                                                       */
14      /*                                                                       */
15      /*     Mods:     Dec_12_91   NRV   Authored.                             */
16      /*                                                                       */
17      /*************************************************************************/
18
19      #define MODES2_C TRUE
20
21      #include "types.h"
22      #include "debug.h"
23      #include <stdio.h>
24      #include "calib.h"
25      #include "modes.h"
26      #include "tm.h"
27      #include "io.h"
28
29
30      /*-----------------------------------------------------------------------*/
31      /*-----------------------------------------------------------------------*/
32      /*-----------------------------------------------------------------------*/
33
34
35
```

```
36          void sfm_std    ()
37          {
38             if (SFM_Mode_Step == 0)
39             {                              /* Set initial grid voltages            */
40                set_grid (1        , -25.0) ;
41                set_grid (2        , +25.0) ;
42                set_grid (3        , -15.0) ;
43                set_va   (SFM_INSTR, -36.0) ;
44             }
45
46             if (SFM_Mode_Step < NUM_PVG_SET_STEPS)
47             {                               /* Set preamp & vernier gain near FS     */
48                set_mode_pvg (SFM_Mode_Step, SFM_INSTR, NEAR_95_FS) ;
49             }
50             else
51             {
52                set_va (SFM_INSTR, -36.0 + 1.0 * (REAL)
53                                            (SFM_Mode_Step - NUM_PVG_SET_STEPS+1)) ;
54             }
55
56          }
```

```
57
58
59
60          void sfm_hi_res ()
61          {
62             if (SFM_Mode_Step == 0)
63             {                                /* Set initial grid voltages            */
64                set_grid (1        , -25.0) ;
65                set_grid (2        , +25.0) ;
66                set_grid (3        , -15.0) ;
67                set_va   (SFM_INSTR, -36.0) ;
68             }
69
70             if (SFM_Mode_Step < NUM_PVG_SET_STEPS)
71             {                                /* Set preamp & vernier gain near FS     */
72                set_mode_pvg (SFM_Mode_Step, SFM_INSTR, NEAR_95_FS) ;
73             }
74             else
75             {
76                set_va (SFM_INSTR, -36.0 + 0.25 * (REAL)
77                                              (SFM_Mode_Step - NUM_PVG_SET_STEPS+1) ) ;
78             }
79          }
```

```
80
81
82
83          void sfm_gt_1 ()
84          {
85             WORD sweep_step ;
86             WORD sweep_num  ;
87
88             sweep_step = SFM_Mode_Step % 45 ;
89             sweep_num  = SFM_Mode_Step / 45 ;
90
91             if (sweep_step == 0)
92             {                              /* Set initial grid voltages            */
93                set_grid (1        , -35.0 + 5.0 * (REAL) sweep_num) ;
94                set_grid (2        , +25.0) ;
95                set_grid (3        , -15.0) ;
96                set_va   (SFM_INSTR, -36.0) ;
97             }
98
99             if (sweep_step < NUM_PVG_SET_STEPS)
100            {                              /* Set preamp & vernier gain near FS     */
101               set_mode_pvg (sweep_step, SFM_INSTR, NEAR_95_FS) ;
102            }
103            else
104            {
105               set_va   (SFM_INSTR, -36.0 - 2.0 * (REAL)
106                                          (sweep_step - NUM_PVG_SET_STEPS+1) ) ;
107            }
108         }
```

```
109
110
111
112         void sfm_gt_2 ()
113         {
114            WORD sweep_step ;
115            WORD sweep_num  ;
116
117            sweep_step = SFM_Mode_Step % 45 ;
118            sweep_num  = SFM_Mode_Step / 45 ;
119
120            if (sweep_step == 0)
121            {                              /* Set initial grid voltages            */
122               set_grid (1        , -25.0) ;
123               set_grid (2        ,  15.0 + 5.0 * (REAL) sweep_num) ;
124               set_grid (3        , -15.0) ;
125               set_va   (SFM_INSTR, -36.0) ;
126            }
127
128            if (sweep_step < NUM_PVG_SET_STEPS)
129            {                              /* Set preamp & vernier gain near FS     */
130               set_mode_pvg (sweep_step, SFM_INSTR, NEAR_95_FS) ;
131            }
132            else
133            {
134               set_va   (SFM_INSTR, -36.0 + 2.0 * (REAL)
135                                          (sweep_step - NUM_PVG_SET_STEPS+1) ) ;
136            }
137         }
```

```
138
139
140
141        void sfm_gt_3 ()
142        {
143           WORD sweep_step ;
144           WORD sweep_num  ;
145
146           sweep_step = SFM_Mode_Step % 45 ;
147           sweep_num  = SFM_Mode_Step / 45 ;
148
149           if (sweep_step == 0)
150           {                              /* Set initial grid voltages            */
151              set_grid (1        , -25.0) ;
152              set_grid (2        , +25.0) ;
153              set_grid (3        , -25.0 + 5.0 * (REAL) sweep_num) ;
154              set_va   (SFM_INSTR, -36.0) ;
155           }
156
157           if (sweep_step < NUM_PVG_SET_STEPS)
158           {                                /* Set preamp & vernier gain near FS     */
159              set_mode_pvg (sweep_step, SFM_INSTR, NEAR_95_FS) ;
160           }
161           else
162           {
163              set_va   (SFM_INSTR, -36.0 + 2.0 * (REAL)
164              .                                (sweep_step - NUM_PVG_SET_STEPS+1) ) ;
165           }
166        }
```

```
167
168
169
170         void sfm_irpa   ()
171         {
172            if (SFM_Mode_Step == 0)
173                                          /* Set initial grid voltages          */
174            {
175               set_grid (1        , -10.0) ;
176               set_grid (2        ,  -5.0) ;
177               set_grid (3        ,   0.0) ;
178               set_va   (SFM_INSTR,   0.0) ;
179            }
180
181            if (SFM_Mode_Step < NUM_PVG_SET_STEPS)
182                                          /* Set preamp & vernier gain near FS  */
183            {
184               set_mode_pvg (SFM_Mode_Step, SFM_INSTR, NEAR_95_FS) ;
185            }
186            else
187            {
188               set_grid (2, -5.0 + 0.2 * (REAL)
189                                          (SFM_Mode_Step - NUM_PVG_SET_STEPS+1) ) ;
190            }
191         }
```

```
190
191
192
193         void sfm_erpa   ()
194         {
195            if (SFM_Mode_Step == 0)
196                                          /* Set initial grid voltages          */
197            {
198               set_grid (1        , +10.0) ;
199               set_grid (2        ,  -5.0) ;
200               set_grid (3        ,   0.0) ;
201               set_va   (SFM_INSTR,   0.0) ;
202            }
203
204            if (SFM_Mode_Step < NUM_PVG_SET_STEPS)
205                                          /* Set preamp & vernier gain near FS  */
206            {
207               set_mode_pvg (SFM_Mode_Step, SFM_INSTR, NEAR_95_FS) ;
208            }
209            else
210            {
211               set_grid (2, -5.0 - 0.2 * (REAL)
212                                          (SFM_Mode_Step - NUM_PVG_SET_STEPS+1) ) ;
213
214            }
215         }
```

```
213
214
215
216        void sfm_clean  ()
217        {
218           if (SFM_Mode_Step == 0)
219           {                                /* Set initial grid voltages          */
220              set_grid (1        , +50.0) ;
221              set_grid (2        , +30.0) ;
222              set_grid (3        , +15.0) ;
223              set_va   (SFM_INSTR,   0.0) ;
224           }
225                                            /* Perform switching to clean probe   */
226           if (SFM_Mode_Step <= 2)
227              clean_prb (SFM_INSTR, SFM_Mode_Step) ;
228                                            /* Perform switching to do measurements */
229           if (SFM_Mode_Step >= CLEANING_TIME+2)
230              uncln_prb (SFM_INSTR, SFM_Mode_Step - (CLEANING_TIME+2)) ;
231        }
```

```
232
233
234
235        void alp_5_pt    ()
236        {
237           REAL i3_real, i4_real ;
238
239           if (ALP_Mode_Step == 0)
240           {
241              set_va_dac (ALP_INSTR, v1_conv) ;
242           }
243           else if (ALP_Mode_Step <= 14)
244           {                                  /* Set preamp & vernier gain to have     */
245                                              /*    A/D input to be near -3.3V         */
246              set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_5PT_VAL) ;
247           }
248           else if (ALP_Mode_Step == 15)
249           {
250              set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_5PT_VAL) ;
251              i1_conv = new_ALP_AD_val ;
252              i1_real = (REAL) new_ALP_AD_val * 0.0048828125 - 10.0 ;
253              v1_conv = ALP_VA_Val ;
254
255              i3_real = -0.471517765 * i1_real ;
256              i3_conv = (WORD) ((i3_real + 10.0) * 204.8) ;
257              vconverge (0, ALP_INSTR, i3_conv) ;
258           }
259           else if ((ALP_Mode_Step >= 16) && (ALP_Mode_Step <= 30))
260           {
261              vconverge (ALP_Mode_Step - 15, ALP_INSTR, i3_conv) ;
262           }
263           else if (ALP_Mode_Step == 31)
264           {
265              vconverge (16, ALP_INSTR, i3_conv) ;
266              v3_conv = ALP_VA_Val ;
267
268              i4_real = -3.0 * i1_real ;
269              i4_conv = (WORD) ((i4_real + 10.0) * 204.8) ;
270              vconverge (0, ALP_INSTR, i4_conv) ;
271           }
272           else if ((ALP_Mode_Step >= 32) && (ALP_Mode_Step <=46))
273           {
274              vconverge (ALP_Mode_Step - 31, ALP_INSTR, i4_conv) ;
275           }
276           else if (ALP_Mode_Step == 47)
277           {
278              vconverge (16, ALP_INSTR, i4_conv) ;
279              v4_conv = ALP_VA_Val ;
280
281              kTe = v4_conv - v3_conv ;
```

```
282                 v2_conv = v1_conv + 2 * kTe ;
283                 set_va_dac (ALP_INSTR, v2_conv) ;
284              }
285              else if (ALP_Mode_Step == 48)
286              {
287                 i2_conv = new_ALP_AD_val ;
288
289                 v5_conv = v4_conv + 7 * (v4_conv - v3_conv) ;
290                 if (v5_conv > ALP_VA_OFFSET + 5 * ALP_VA_GAIN)
291                    v5_conv = ALP_VA_OFFSET + 5 * ALP_VA_GAIN ;
292                 set_va_dac (ALP_INSTR, v5_conv) ;
293              }
294              else if (ALP_Mode_Step == 49)
295              {
296                 i5_conv = new_ALP_AD_val ;
297
298                 v1_conv = ALP_VA_OFFSET - 9 * kTe ;
299                 if (v1_conv < ALP_VA_OFFSET - 5 * ALP_VA_GAIN)
300                    v1_conv = ALP_VA_OFFSET - 5 * ALP_VA_GAIN ;
301              }
302           }
```

```
303
304
305
306
307        void alp_std    ()
308        {
309           if (ALP_Mode_Step == 0)
310           {
311              set_va (ALP_INSTR, 5.0) ;
312           }
313
314           if (ALP_Mode_Step < NUM_PVG_SET_STEPS)
315                                        /* Set preamp & vernier gain near FS     */
316           {
317              set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_95_FS) ;
318           }
319           else
320           {
321              set_va (ALP_INSTR, 5.0 - 0.2 * (REAL)
322                                        (ALP_Mode_Step - NUM_PVG_SET_STEPS+1) ) ;
323
           }
        }
```

```
324
325
326
327        void alp_hi_res ()
328        {
329           if (ALP_Mode_Step == 0)
330           {
331              set_va (ALP_INSTR, 1.7) ;
332           }
333
334           if (ALP_Mode_Step < NUM_PVG_SET_STEPS)
335                                        /* Set preamp & vernier gain near FS     */
336           {
337              set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_95_FS) ;
338           }
339           else
340           {
341              set_va (ALP_INSTR, 1.7 - 0.04 * (REAL)
342                                        (ALP_Mode_Step - NUM_PVG_SET_STEPS) ) ;
343
           }
        }
```

```
344
345
346
347     void alp_m_res ()
348     {
349        if (ALP_Mode_Step == 0)
350        {
351           set_va (ALP_INSTR, ALP_VA_GAIN * (REAL) kTe + 2.0) ;
352        }
353
354        if (ALP_Mode_Step < NUM_PVG_SET_STEPS)
355                                        /* Set preamp & vernier gain near FS     */
356        {
357           set_mode_pvg (ALP_Mode_Step, ALP_INSTR, NEAR_95_FS) ;
358        }
359        else
360        {
361           set_va_dac (ALP_INSTR, ALP_VA_OFFSET + kTe - (kTe / 8) *
362                                        (ALP_Mode_Step - NUM_PVG_SET_STEPS) ) ;
363
        }
     }
```

SFMALP.ASM

```
                    ;                 ***************************************************
                    ;                 *                                                 *
                    ;                 *                                                 *
                    ;                 *                   E Q U A T E S                 *
                    ;                 *                                                 *
                    ;                 *                                                 *
                    ;                 ***************************************************
                    ;
                                                  ;   Serial Flag Register bit locations
0000                Xmt_Active:     equ     0       ; Bit 0 = Xmt Active
0001                Xmt_Full:       equ     1       ; Bit 1 = Xmt Full
0002                Xmt_Empty:      equ     2       ; Bit 2 = Xmt Empty
0003                Rcv_Full:       equ     3       ; Bit 3 = Rcv Full
0004                Rcv_Empty:      equ     4       ; Bit 4 = Rcv Empty
0005                CR_Detected:    equ     5       ; Bit 5 = CR Detected
                                                    ; Bit 6 =
                                                    ; Bit 7 =

                                                  ;   Flags register assignment
0000                Timer1_Int:     equ     0       ; Bit 0 = Timer 1 Interrupt Occured
0001                Jiffy_Override: equ     1       ; Bit 1 = Software Enable of Jiffy Timer
                                                    ; Bit 2 =
                                                    ; Bit 3 =
                                                    ; Bit 4 =
                                                    ; Bit 5 =
                                                    ; Bit 6 =
                                                    ; Bit 7 =

                                                  ; Flags 2 Register assignment
                                                    ; Bit 0 =
                                                    ; Bit 1 =
                                                    ; Bit 2 =
                                                    ; Bit 3 =
                                                    ; Bit 4 =
                                                    ; Bit 5 =
                                                    ; Bit 6 =
                                                    ; Bit 7 =

                                                  ;  IO_PORT2 pin assignments (input port)
                                                    ; Bit 0 =
                                                    ; Bit 1 =
                                                    ; Bit 2 =
0003                Accel:          equ     3       ; Bit 3 =
```

```
0004        T_Full:     equ     4       ; Bit 4 =
                                        ; Bit 5 =
                                        ; Bit 6 =
                                        ; Bit 7 =


                                    ; ASCII characters
000D        Cr:         equ     13      ; Carriage Return
000A        Lf:         equ     10      ; Line Feed
0008        Bs:         equ     08      ; Back Space
007F        Del:        equ     7Fh     ; Delete
0007        Bel:        equ     07      ; Bell
000C        FF:         equ     12      ; Form Feed
0009        Ht:         equ     9       ; Horizontal Tab
000B        Vt:         equ     11      ; Vertical Tab
0020        Sp:         equ     32      ; Space

0000        Port1_init:         equ     000h    ; No Instrument Selected
00FF        Port2_init:         equ     0FFh    ; All Inputs HIGH
```

298

```
                    ;               ***************************************************
                    ;               *                                                 *
                    ;               *                                                 *
                    ;               *      E X T E R N A L   A D D R E S S E S      *
                    ;               *                                                 *
                    ;               *                                                 *
                    ;               ***************************************************
                    ;

2100                loader:         equ     02100h          ; Loader Entry Point
4000                Loader_CS:      equ     04000h          ; Loader Checksum

C000                ZRAM_start:     equ     0C000h          ; Start address of ZRAM
DFFF                ZRAM_stop:      equ     0DfffH          ; Stop  address of ZRAM

4000                EEPROM_start    equ     04000h          ; Start address of EEPROM
C000                EEPROM_stop     equ     0C000h          ; Stop  address of EEPROM

F000                Grid_Volt_WR:   equ     0F000h          ; Grid Voltage DAC Write
F000                Timer_WR:       equ     0F000h          ; Programmable Timer Write
F000                Probe_Cln_WR:   equ     0F000h          ; Cleaning Control Word Write
F400                EMeter_RD:      equ     0F400h          ; Electrometer A/D Read
F800                VAGen_WR:       equ     0F800h          ; VA Generator DAC Write
F800                EMeter_WR:      equ     0F800h          ; Electrometer Preamp & Vernier Gain
FC00                Telem_WR:       equ     0FC00h          ; Write to telemetry FIFO
FC00                Telem:          equ     0FC00h          ; Write to telemetry FIFO
```

```
                    ;          ************************************************
                    ;          *                                              *
                    ;          *                                              *
                    ;          *          S T A C K   S E G M E N T S         *
                    ;          *                                              *
                    ;          *                                              *
                    ;          *                                              *
                    ;          ************************************************
                    ;
                                RSEG    CSTACK(1)

0000                            DSB     2048            ; C program stack
0800                Stack_Ptr:
```

```
                    ;                 ***************************************************
                    ;                 *                                                 *
                    ;                 *                                                 *
                    ;                 *                                                 *
                    ;                 *          Z E R O   P O W E R   R A M            *
                    ;                 *                                                 *
                    ;                 *                                                 *
                    ;                 *                S E G M E N T                    *
                    ;                 *                                                 *
                    ;                 ***************************************************


                    ;                 Zero Power Ram Locations


0000                          RSEG    ZRAM

                    ;                 Misc Data Storage, Global To All Program Modes

                    Rom_CS:           DSB     2                          ; Rom checksum


0002 0000           ALP_Mode_Ptr:     DCW     0000
0004 0000           SFM_Mode_Ptr:     DCW     0000
0006 0000           Curr_ALP_Mode:    DCW     0000
0008 0000           Curr_SFM_Mode:    DCW     0000
000A 0000           ALP_Mode_Step:    DCW     0000
000C 0000           SFM_Mode_Step:    DCW     0000
000E 0000           Clock_ZP:         DCW     0000
0010 0000                             DCW     0000

0012 0000           Status:           DCW     0000

0014 0000           SFM_Vern_PG:      DCW     0000
0016 0000           ALP_Vern_PG:      DCW     0000
0018 401F           SFM_VA_Val:       DCW     8000
001A 401F           ALP_VA_Val:       DCW     8000

001C 0000           clean_word:       DCW     0000

001E                ALP_mode_start_time:    DSB     4
0022                SFM_mode_start_time:    DSB     4

0026                Rcv_Buffer:       DSB     80d                        ; Recieve buffer
0076                Rcv_End:
0076                Xmt_Buffer:       DSB     220d                       ; Transmit buffer
0152                Xmt_End:

0000                          RSEG    Telem_Buff
```

```
400  0000        ALP_telem:     DSB    768d
401  0300        SFM_telem:     DSB    768d
402  0600        HSKP_telem:    DSB     64d
403  0640        IDLE_telem:    DSB     32d
404
```

```
                ;               ************************************************
                ;               *                                              *
                ;               *            C O D E   S E G M E N T           *
                ;               *                                              *
                ;               *                      &                       *
                ;               *                                              *
                ;               *       R E S E T   E N T R Y   P O I N T      *
                ;               *                                              *
                ;               ************************************************
                ;

                ;               Code Reset Entry Point

0000                    RSEG    ROM(2)

0000 A1000818   Reset:  ld      SP, #Stack_Ptr          ; set stack pointer to top of RAM.
                                                        ;     stack moves down.
0004 1114               clrb    WSR                     ; *** 80c196 register ***
                                                        ; 80c196 Window Select Register.
0006 B15503             ldb     HSI_MODE, #01010101b    ; all HSI responds to rising edges
0009 B1A015             ldb     IOC0, #10100000b        ; HSI.0          = disabled
                                                        ; T2 rst on /WR  = disabled
                                                        ; HSI.1          = disabled
                                                        ; T2 ext rst     = disabled
                                                        ; HSI.2          = disabled
                                                        ; T2 rst         = HSI.0
                                                        ; HSI.3          = disabled
                                                        ; T2clk          = HSI.1
000C B1A116             ldb     IOC1, #10100001b        ; PWM            = enabled
                                                        ; ExtInt         = ExtInt pin
                                                        ; T1 ovrflo int  = disabled
                                                        ; T2 ovrflo int  = disabled
                                                        ; HSO.4          = disabled
                                                        ; TXD            = enabled
                                                        ; HSO.5          = disabled
                                                        ; HSI Interrupt  = Fifo Full
000F B18017             ldb     IOC2, #10000000b        ; *** 80c196 registers ***
                                                        ; Clear CAM      = Yes
                                                        ; Cam Lock       = No
                                                        ; T2 alt int     = No
                                                        ; AD prescale    = Normal
                                                        ; AD sample/hold= Enabled
                                                        ; Slow PWM       = No
                                                        ; T2 U/D ena     = No
                                                        ; Fast T2 ena    = No

0012 FA                 di                              ; disable interrupts
```

```
0013 1108          clrb   INT_MASK              ; mask all interrupts
0015 1113          clrb   INT_MASK1             ; *** 80c196c register ***
0017 1109          clrb   INT_PEND              ; clear any pending interrupts
0019 1112          clrb   INT_PEND1             ; *** 80ci96 register ***
001B 1131          clrb   Flags                 ; clear General Purpose Flag register
001D 1132          clrb   Flags2                ;
001F B1000F        ldb    IO_PORT1, #Port1_init ; init Port 1
0022 B1FF10        ldb    IO_PORT2, #Port2_init ; init Port 2


0025 EFFFFE  Rst_10: lcall  ?SEG_INIT_LOO       ; initialize 'C' code
0028 EFFFFE          call   Init_Serial         ; initialize serial port

002B FB              ei                         ; enable interrupts
002C 111A            clrb   A1
002E EFFFFE          call   Init_Timer1         ; initialize 20mS clock
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                          M A I N   P R O G R AM                          *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |   outputs    |   routines called    |   registers altered  *
                ;*****************************************************************************
                ;*   none      |              |       Xmt_Char       |         Flags        *
                ;*             |              |       Rcv_Char       |                      *
                ;*****************************************************************************

                ;               This is the lowest level infinite loop.  This routine
                ;       basically sits in a wait loop until a timer interrupt causes
                ;       reading of the CCD chip or a serial interrupt occures.
                ;
                ;               Upon receipt of a carriage return the 'C' code routine
                ;       'main' is executed to handle the user interface.  The stack pointer
                ;       is switched to the external RAM 'C' code stack and reading of the
                ;       CCD chip is disabled.
                ;               Returning from the 'C' code returns the stack pointer to
                ;       internal RAM and re-enables the reading of the CCD chip.

0031            Main_Program:
0031 EFFFFE             lcall   main                            ; call 'C' code
0034 27FB               sjmp    Main_Program                    ; loop forever
```

```
503
504                 ;*****************************************************************************
505                 ;*                                                                           *
506                 ;*                                                                           *
507                 ;*                                  D E L A Y                                *
508                 ;*                                                                           *
509                 ;*                                                                           *
510                 ;*****************************************************************************
511                 ;* inputs      |    outputs    |    routines called    |   registers altered *
512                 ;*****************************************************************************
513                 ;*     Ax      |      none     |          none         |          Ax         *
514                 ;*****************************************************************************
515
516                 ;               The number of milliseconds to delay is passed in Ax.  Delay
517                 ;       is accomplished with a wait loop.
518                 ;
519                 ;               delay() can be called from 'C' code which passes an integer
520                 ;       delay period in milliseconds.
521
522  014C           Delay_Const_96  equ     332             ; 1 mSec constant 8096
523  01F2           Delay_Const_196 equ     498             ; 1 mSec constant 80c196
524
525
526
527  0036 A318021A  delay:  ld      Ax, 2 [SP]              ; read 'C' passed delay period
528
529  003A C81C      Delay:  push    Bx                      ; save Bx & Cx register
530
531  003C A1F2011C  D_30:   ld      Bx, #Delay_Const_196    ; Pgm in 1 mSec delay constant
532  0040 051C      D_40:   dec     Bx
533  0042 D7FC              jne     D_40
534
535  0044 051A              dec     Ax
536  0046 D7F4              jne     D_30
537
538  0048 CC1C              pop     Bx
539  004A F0                ret
540
```

```
542
543                 ;*****************************************************************************
544                 ;*                                                                           *
545                 ;*                                                                           *
546                 ;*                              S L E E P                                    *
547                 ;*                                                                           *
548                 ;*                                                                           *
549                 ;*****************************************************************************
550                 ;* inputs      |    outputs    |    routines called    |   registers altered *
551                 ;*****************************************************************************
552                 ;*  Flags      |               |                       |                     *
553                 ;*             |               |                       |                     *
554                 ;*****************************************************************************
555
556                 ;       This routine waits until a Timer1 interrupt is generated.  In the
557                 ;       case of the 80C196 processor the CPU will be put into a low power
558                 ;       sleep mode.
559
560  004B           sleep:
561                         ; idlpd   #1                        ; 80C196 sleep instruction
562  004B 3031FD            jbc     Flags,Timer1_Int,sleep  ; wait for Timer interrupt
563  004E 71FE31            andb    Flags, #11111110b       ; clear Timer1_Int flag
564  0051 F0                ret
565
```

```
567
568                 ;*****************************************************************************
569                 ;*                                                                           *
570                 ;*                                                                           *
571                 ;*                          R E A D    A D                                   *
572                 ;*                                                                           *
573                 ;*                                                                           *
574                 ;*****************************************************************************
575                 ;* inputs      |    outputs    |    routines called    |   registers altered *
576                 ;*****************************************************************************
577                 ;*    SP       |      R0       |                       |                     *
578                 ;*             |               |                       |                     *
579                 ;*****************************************************************************
580
581                 ;              This routine recieves a channel number then performs an
582                 ;       A/D conversion and returns the results.
583
584  0052           read_AD:
585
586  0052 A318021A          ld      Ax, 2[SP]              ; get channel number off stack
587  0056 91081A            orb     A1, #00001000b         ; set go flag to start A/D
588  0059 B01A02            ldb     AD_COMMAND, A1         ; send A/D command
589
590  005C C81C              push    Bx                     ; save Bx
591  005E 011C              clr     Bx                     ; clear result register
592  0060 0000              skip    0                      ; short delay for conversion
593  0062 0000              skip    0                      ;   to begin
594  0064 0000              skip    0
595  0066 0000              skip    0
596  0068 0000              skip    0
597
598  006A B0021A    RA_10:  ldb     A1, AD_RESULT_LO       ; read A/D result register
599  006D 3B1AFA            jbs     A1, 3, RA_10           ; Wait for conversion complete
600  0070 B0031C            ldb     B1, AD_RESULT_HI       ; read high byte of A/D conversion
601  0073 09021C            shl     Bx, #2                 ; shift Bx up two bits
602  0076 18061A            shrb    A1, #6                 ; shift A1 down 6 bits
603  0079 901A1C            orb     B1, A1                 ; combine
604  007C A01C1C            ld      R0, Bx                 ; place result into R0
605  007F CC1C              pop     Bx
606  0081 F0                ret
607
608  0082                   END
```

```
Errors:  None        ##########
Bytes:   158         # sfmalp #
CRC:     70D0        ##########
```

Symbol and Cross Reference Table

Symbol	Value	Type	Defline	Refline
Segment Definitions				
CSTACK	0800	S00R	345	
ROM	0082	S03R	421	
Telem_Buff	0660	S02R	398	
ZRAM	0152	S01R	365	
External Symbols				
?SEG_INIT_L00	0000	X01	85	465
Asc_Hex	0000	X0A	90	
Init_Serial	0000	X07	89	466
Init_Timer1	0000	X0D	94	470
Rcv_Char	0000	X02	89	
Rx_Int	0000	X05	89	
Send_Hex_Byte	0000	X09	90	
Send_Hex_Word	0000	X08	90	
Send_Msg	0000	X04	89	
Tx_int	0000	X06	89	
Wait_CR	0000	X0B	90	
Wait_Tx_Inactive				

```
          0000  X0C     90

Xmt_Char  0000  X03     89

main      0000  X00     85      499
```

Public Symbols
==========

```
ALP_Mode_Ptr
          0002  R01E    372      68

ALP_Mode_Step
          000A  R01E    376      68

ALP_VA_Val
          001A  R01E    386      47

ALP_Vern_PG
          0016  R01E    384      47

ALP_mode_start_time
          001E  R01E    390      70

ALP_telem
          0000  R02E    400      68

Ah        001B  AE#     195      44

Al        001A  AE#     196      44  469  587  588  598  599  602
                                 603

Ax        001A  AE#     194      44  527  535  586

Bl        001C  AE#     199      44  600  603

Bx        001C  AE#     197      44  529  531  532  538  590  591
                                 601  604  605

Ch        001F  AE#     201      44

Cl        001E  AE#     202      44

Clock_ZP  000E  R01E    378      48

Curr_ALP_Mode
          0006  R01E    374      68
```

Curr_SFM_Mode	0008	R01E	375	69			
Cx	001E	AE#	200	44			
Delay	003A	R03E	529	59			
Dh	0021	AE#	204	44			
Dl	0020	AE#	205	44			
Dx	0020	AE#	203	44			
EEPROM_start	4000	AE	315	53			
EEPROM_stop	C000	AE	316	53			
EMeter_RD	F400	AE	321	62			
EMeter_WR	F800	AE	323	62			
Eh	0023	AE#	207	45			
El	0022	AE#	208	45			
Ex	0022	AE#	206	45			
Fh	0025	AE#	210	45			
Fl	0024	AE#	211	45			
Flags	0031	AE#	224	46	459	562	563
Flags2	0032	AE#	225	46	460		
Fx	0024	AE#	209	45			
Grid_Volt_WR	F000	AE	318	62			
HSKP_telem	0600	R02E	402	72			
IDLE_telem	0640	R02E	403	72			

Loader_CS	4000	AE	310	53
Probe_Cln_WR	F000	AE	320	63
Rcv_Buffer	0026	R01E	393	67
Rcv_End	0076	R01E	394	67
Rom_CS	0000	R01E	369	53
SFM_Mode_Ptr	0004	R01E	373	69
SFM_Mode_Step	000C	R01E	377	69
SFM_VA_Val	0018	R01E	385	47
SFM_Vern_PG	0014	R01E	383	47
SFM_mode_start_time	0022	R01E	391	70
SFM_telem	0300	R02E	401	69
Serial_Flags	0030	AE#	223	46
Status	0012	R01E	381	46
Telem	FC00	AE	325	63
Telem_WR	FC00	AE	324	63
Timer_WR	F000	AE	319	62
VAGen_WR	F800	AE	322	62
Xmt_Buffer	0076	R01E	395	67
Xmt_End	0152	R01E	396	67

ZRAM_start	C000	AE	312	52	
ZRAM_stop	DFFF	AE	313	52	
_Bh	001D	AE#	198	44	
clean_word	001C	R01E	388	71	
delay	0036	R03E	527	56	
loader	2100	AE	309	56	
read_AD	0052	R03E	584	56	
sleep	004B	R03E	560	56	562

Local Symbols
==========

AD_COMMAND	0002	A#	137	588
AD_RESULT_HI	0003	A#	136	600
AD_RESULT_LO	0002	A#	135	598
Accel	0003	A	276	
BAUD_RATE	000E	A#	158	
Bel	0007	A	288	
Bs	0008	A	286	
CR_Detected	0005	A	245	
Cr	000D	A	284	
D_30	003C	R03	531	536

D_40	0040	R03	532	533
Del	007F	A	287	
Delay_Const_196	01F2	A	523	531
Delay_Const_96	014C	A	522	
FF	000C	A	289	
HSI_MODE	0003	A#	139	427
HSI_STATUS	0006	A#	141	
HSI_TIME	0004	A#	140	
HSO_COMMAND	0006	A#	144	
HSO_TIME	0004	A#	143	
Ht	0009	A	290	
INT_MASK	0008	A#	150	455
INT_MASK1	0013	A#	188	456
INT_PEND	0009	A#	151	457
INT_PEND1	0012	A#	187	458
IOC0	0015	A#	170	428
IOC1	0016	A#	171	436
IOC2	0017	A#	191	444
IOS0	0015	A#	167	
IOS1	0016	A#	168	
IOS2	0017	A#	190	
IO_PORT0	000E	A#	160	

)RT1	000F	A#	161	461							
)RT2	0010	A#	162	462							
ASE	001A	A#	115	117	118	120	121	122	123	124	
				125	126	127					
y_Clock											
	002E	A#	221								
y_Override											
	0001	A	252								
	000A	A	285								
ı_Program											
	0031	R03	498	500							
_CONTROL											
	0017	A#	173								
:1_init											
	0000	A	294	461							
:2_											
	00FF	A	295	462							
	001C	A#	120	604							
	001D	A#	121								
	001E	A#	122								
	001F	A#	123								
	0020	A#	124								
	0021	A#	125								
	0022	A#	126								
	0023	A#	127								
	001A	A#	117								
	001B	A#	118								
_10	006A	R03	598	599							

Rcv_Empty	0004	A	244			
Rcv_Full	0003	A	243			
Rcv_In_Ptr	0026	A#	216			
Rcv_Out_Ptr	0028	A#	217			
Reset	0000	R03	423			
Rst_10	0025	R03	465			
SBUF	0007	A#	146			
SBUF_RX	0007	A#	147			
SBUF_TX	0007	A#	148			
SP	0018	A#	132	423	527	586
SP_CON	0011	A#	165			
SP_STAT	0011	A#	164			
STACKPOINTER	0018	A#	133			
Sp	0020	A	292			
Stack_Ptr	0800	R00	348	423		
TIMER1	000A	A#	155			
TIMER2	000C	A#	156			
T_Full	0004	A	277			
Timer1_Int	0000	A	251	562		
Vt	000B	A	291			
WATCHDOG	000A	A#	153			

WSR	0014	A#	189	425
Xmt_Active	0000	A	240	
Xmt_Empty	0002	A	242	
Xmt_Full	0001	A	241	
Xmt_In_Ptr	002A	A#	218	
Xmt_Out_Ptr	002C	A#	219	
ZERO	0000	A#	213	
banking	0000	A#	181	

Macro Definitions
==========

ISR.ASM

```
#################################################################
#                                                               #
#     Archimedes 8096/196 Assembler V1.95/MD2        13/Jul/92  04:03:47   #
#                                                               #
#      Source   =  isr.asm                                      #
#      List     =  isr.lst                                      #
#      Object   =  isr.obj                                      #
#      Options  =  s                                            #
#                                                               #
#                          (c) Copyright Archimedes Software Inc. 1988 #
#################################################################
```

```
                ;*********************************************************************
                ;*                                                                   *
                ;*                  S O L A R   F L U X   M O N I T O R              *
                ;*                                                                   *
                ;*             A D V A N C E D   L A N G M U I R   P R O B E         *
                ;*                                                                   *
                ;*                          S F M / A L P                            *
                ;*                                                                   *
                ;*                                                                   *
                ;*                     Interrupt Service Routines                    *
                ;*                                                                   *
                ;*********************************************************************


0000                    Name    ISR
0000                    Rseg    ISR


                ;                       Assembler Directives
0000                    LSTCOD  +
0000                    LSTWID  -
0000                    LSTPAG  +
0000                    LSTXRF
0000                    PAGSIZ  52
```

32
33
34

```
;               ***************************************************
;               *                                                 *
;               *                                                 *
;               *                                                 *
;               *                  P U B L I C S                  *
;               *                                                 *
;               *                                                 *
;               ***************************************************
```

```
                ;               ***************************************************
                ;               *                                                 *
                ;               *                                                 *
                ;               *                                                 *
                ;               *              E X T E R N A L S                  *
                ;               *                                                 *
                ;               *                                                 *
                ;               ***************************************************

                ;
                ;               Serial Externals

0000            Extern  Rcv_Char, Xmt_Char, Send_Msg, Rx_Int, Tx_int, Init_Serial
0000            Extern  Send_Hex_Word, Send_Hex_Byte, Asc_Hex, Wait_CR, Wait_Tx_Inactive

                ;               Timer Externals

0000            Extern  Timer1, Timer2
```

```
                ;
                ;               ***************************************************
                ;               *                                                 *
                ;               *      R E G I S T E R   D E F I N I T I O N S    *
                ;               *                                                 *
                ;               ***************************************************

                $REG8096.INC
                ;       - REG8096.INC -         ;INCLUDE FILE FOR 8096 REGISTERS
                ;
                ;       Include file containing 8096 register definitions
                ;

                ;
                ;       8096 C Compiler Working registers
                ;
                IR_BASE DEFINE  01AH                    ; Register base address

 001A           R8      DEFINE  (IR_BASE)
 001B           R9      DEFINE  (IR_BASE+1)

 001C           R0      DEFINE  (IR_BASE+2)
 001D           R1      DEFINE  (IR_BASE+3)
 001E           R2      DEFINE  (IR_BASE+4)
 001F           R3      DEFINE  (IR_BASE+5)
 0020           R4      DEFINE  (IR_BASE+6)
 0021           R5      DEFINE  (IR_BASE+7)
 0022           R6      DEFINE  (IR_BASE+8)
 0023           R7      DEFINE  (IR_BASE+9)

                ;
                ;       Special Function Registers
                ;
 0018           SP              DEFINE  18H
 0018           STACKPOINTER    DEFINE  18H

 0002           AD_RESULT_LO    DEFINE  02H
 0003           AD_RESULT_HI    DEFINE  03H
 0002           AD_COMMAND      DEFINE  02H

 0003           HSI_MODE        DEFINE  03H
 0004           HSI_TIME        DEFINE  04H
 0006           HSI_STATUS      DEFINE  06H

 0004           HSO_TIME        DEFINE  04H
 0006           HSO_COMMAND     DEFINE  06H
```

```
0007        SBUF          DEFINE  07H
0007        SBUF_RX       DEFINE  07H
0007        SBUF_TX       DEFINE  07H

0008        INT_MASK      DEFINE  08H
0009        INT_PEND      DEFINE  09H

000A        WATCHDOG      DEFINE  0AH

000A        TIMER1        DEFINE  0AH
000C        TIMER2        DEFINE  0CH

000E        BAUD_RATE     DEFINE  0EH

000E        IO_PORT0      DEFINE  0EH
000F        IO_PORT1      DEFINE  0FH
0010        IO_PORT2      DEFINE  10H

0011        SP_STAT       DEFINE  11H
0011        SP_CON        DEFINE  11H

0015        IOS0          DEFINE  15H
0016        IOS1          DEFINE  16H

0015        IOC0          DEFINE  15H
0016        IOC1          DEFINE  16H

0017        PWM_CONTROL   DEFINE  17H

            $DEF8096.INC
            ;       - DEF8096.INC -
            ;
            ;       Include file for getting a nonbanked model
            ;

0000        banking       DEFINE  0


            $SFMALP.INC

0012        INT_PEND1     DEFINE    ; 80C196 Special Function Registers
0013        INT_MASK1     DEFINE    12h
0014        WSR           DEFINE    13h
0017        IOS2          DEFINE    14h
0017        IOC2          DEFINE    17h
                                    17h

001A        Ax            DEFINE    ; Registers to be used in assembly routines
                                    1Ah
```

```
001B        Ah          DEFINE      1Bh
001A        Al          DEFINE      1Ah
001C        Bx          DEFINE      1Ch
001D        _Bh         DEFINE      1Dh
001C        Bl          DEFINE      1Ch
001E        Cx          DEFINE      1Eh
001F        Ch          DEFINE      1Fh
001E        Cl          DEFINE      1Eh
0020        Dx          DEFINE      20h
0021        Dh          DEFINE      21h
0020        Dl          DEFINE      20h
0022        Ex          DEFINE      22h
0023        Eh          DEFINE      23h
0022        El          DEFINE      22h
0024        Fx          DEFINE      24h
0025        Fh          DEFINE      25h
0024        Fl          DEFINE      24h

0000        ZERO        DEFINE      00h ; To be used in comparisons

                                    ; Serial Port, Jiffly Clock, and Flags
0026        Rcv_In_Ptr  DEFINE      26h
0028        Rcv_Out_Ptr DEFINE      28h
002A        Xmt_In_Ptr  DEFINE      2Ah
002C        Xmt_Out_Ptr DEFINE      2Ch

002E        Jiffy_Clock DEFINE      2Eh

0030        Serial_Flags DEFINE     30h
0031        Flags       DEFINE      31h
0032        Flags2      DEFINE      32h
```

```
                ;       ************************************************
                ;       *                                              *
                ;       *                                              *
                ;       *                EQUATES                       *
                ;       *                                              *
                ;       *                                              *
                ;       ************************************************



0000                                  ;  Serial Flag Register bit locations
0001   Xmt_Active:       equu   0         ; Bit 0 = Xmt Active
0002   Xmt_Full:         equ    1         ; Bit 1 = Xmt Full
0003   Xmt_Empty:        equ    2         ; Bit 2 = Xmt Empty
0004   Rcv_Full:         equ    3         ; Bit 3 = Rcv Full
0005   Rcv_Empty:        equ    4         ; Bit 4 = Rcv Empty
       CR_Detected:      equ    5         ; Bit 5 = CR Detected
                                          ; Bit 6 =
                                          ; Bit 7 =

0000                                  ;  Flags register assignment
0001   Timer1_Int:       equ    0         ; Bit 0 = Timer 1 Interrupt Occured
       Jiffy_Override:   equ    1         ; Bit 1 = Software Enable of Jiffy Timer
                                          ; Bit 2 =
                                          ; Bit 3 =
                                          ; Bit 4 =
                                          ; Bit 5 =
                                          ; Bit 6 =
                                          ; Bit 7 =


                                      ; Flags 2 Register assignment
                                          ; Bit 0 =
                                          ; Bit 1 =
                                          ; Bit 2 =
                                          ; Bit 3 =
                                          ; Bit 4 =
                                          ; Bit 5 =
                                          ; Bit 6 =
                                          ; Bit 7 =


                                      ;  IO_PORT2 pin assignments (input port)
                                          ; Bit 0 =
0003   Accel:            equ    3         ; Bit 1 =
                                          ; Bit 2 =
                                          ; Bit 3 =
```

```
0004        T_Full:     equ     4       ; Bit 4 =
                                        ; Bit 5 =
                                        ; Bit 6 =
                                        ; Bit 7 =


                                ; ASCII characters
000D        Cr:         equ     13      ; Carriage Return
000A        Lf:         equ     10      ; Line Feed
0008        Bs:         equ     08      ; Back Space
007F        Del:        equ     7Fh     ; Delete
0007        Bel:        equ     07      ; Bell
000C        FF:         equ     12      ; Form Feed
0009        Ht:         equ     9       ; Horizontal Tab
000B        Vt:         equ     11      ; Vertical Tab
0020        Sp:         equ     32      ; Space

0000        Port1_init:         equ     000h    ; No Instrument Selected
00FF        Port2_init:         equ     0FFh    ; All Inputs HIGH
```

267

```
                ;               ***************************************************
                ;               *                                                 *
                ;               *                                                 *
                ;               *     C H E C K S U M ,   L O A D E R   K E Y     *
                ;               *                                                 *
                ;               *                       &                         *
                ;               *                                                 *
                ;               *        R E S E T   E N T R Y   P O I N T        *
                ;               *                                                 *
                ;               ***************************************************


0000                    RSEG    LOAD_ISR(2)
0000            Rom_Start:

0000            CS:             dsb         2           ; EEPROM Checksum \ Loader Generated
0002 5026472E   Key:            dcb   'P&G.Inc '        ; Loader Key
0006 496E6320

                                                        ; Software Version Date
000A 0700       Month:          dcw     Date 5          ;    Version date calculated by
000C 0D00       Day:            dcw     Date 4          ;    by assembler
000E 5C00       Year:           dcw     Date 6
```

```
294
295
296                 ;*****************************************************************************
297                 ;*                                                                           *
298                 ;*                                                                           *
299                 ;*                    I N T E R R U P T    S U B R O U T I N E S              *
300                 ;*                                                                           *
301                 ;*                                                                           *
302                 ;*****************************************************************************
303
304                                    org     Rom_Start + 10h
305  0010
306
307  0010 E72A02    T_Overflow:        ljmp    Error_Int_1     ; Timer Overflow ISR
308                                    org     Rom_Start + 20h
309  0020
310
311  0020 E72102    AD_Complete:       ljmp    Error_Int_2     ; A/D Conversion Complete ISR
312                                    org     Rom_Start + 30h
313  0030
314
315  0030 E71802    HSI_Available:     ljmp    Error_Int_3     ; HSI Data Available vector
316                                    org     Rom_Start + 40h
317  0040
318
319  0040 E7BD00    HSO:               ljmp    Software_Timer_ISR ; High speed Output vector
320                                    org     Rom_Start + 50h
321  0050
322
323  0050 E70602    HSI1:              ljmp    Error_Int_5     ; HSI.1 external interrupt vector
324                                    org     Rom_Start + 60h
325  0060
326
327  0060 E7FD01    Trap_Vector:       ljmp    Error_Int_6     ; Trap vector
328                                    org     Rom_Start + 70h
329  0070
330
331  0070 E7F401    OpCode:            ljmp    Error_Int_7     ; Unimplimented Opcode vector
332                                    org     Rom_Start + 80h
333  0080
334
335  0080 E7EB01    HSI4:              ljmp    Error_Int_8     ; 4th HSI FIFO entry vector
336                                    org     Rom_Start + 90h
337  0090
338
339  0090 E7E201    T2cap:             ljmp    Error_Int_9     ; Timer #2 Capture vector
340                                    org     Rom_Start + 0A0h
341  00A0
342
```

```
00A0 E7D901    T2overflow:    ljmp    Error_Int_A     ; Timer #2 Overflow vector

                              org     Rom_Start + 0B0h
00B0

00B0 E7D001    ExInt_Pin:     ljmp    Error_Int_B     ; External interrupt pin vector

                              org     Rom_Start + 0C0h
00C0

00C0 E7C701    HSI_Full:      ljmp    Error_Int_C     ; HSI FIFO Full Vector

                              org     Rom_Start + 0D0h
00D0

00D0 E7BE01    NMI:           ljmp    Error_Int_D     ; NMI vectors
```

```
359                 ;*****************************************************************************
360                 ;*                                                                           *
361                 ;*                          S O F T W A R E   T I M E R   I S R              *
362                 ;*                                                                           *
363                 ;*****************************************************************************
364                 ;*                                                                           *
365                 ;* inputs     |   outputs    |   routines called   |   registers altered   *
366                 ;*****************************************************************************
367                 ;*   none     |     none     |        Timer1       |       INT_PEND        *
368                 ;*****************************************************************************
369
370                 ;       This is the Software timer interrupt sub routine.  It
371                 ;       determines which software timer generated an interrupt and calls
372                 ;       its interrupt handler.  It is possible for more than one software
373                 ;       timer to generate an interrupt at the same time.
374                 ;               Timer 1 time is passed to the individual routines in Ax while
375                 ;       the HSO status is passed in Bl.  Each Timer handling routine must
376                 ;       be written like an interrupt sub routine, no registers that may effect
377                 ;       other routines may be altered or corrupted.
378                 ;               Lastly, when a software interrupt is generated, a HSO interrupt
379                 ;       may also be generated.  This routine will cancel any pending HSO
380                 ;       interrupt upon its exit.
381
382
383                         org     Rom_Start + 100h
384  0100
385
386  0100           Software_Timer_ISR:                                ; save PSW, disable all interrupts
387                         pusha
388  0100 F4                push    Ax                                 ; read present time into Ax. 2.5 timer
389  0101 C81A              ld      Ax, TIMER1                         ;    states have so far elapsed.
390  0103 A00A1A                                                       ; eliminate concurrent HSO interrupt
391                         andb    INT_PEND, #11110111b
392  0106 71F709            push    Bx
393  0109 C81C              push    Cx
394  010B C81E              push    Dx
395  010D C820              orb     INT_MASK, #01100000b               ; serial, timer interrupts
396  010F 916008            ei                                         ; re-enable interrupts
397  0112 FB
398                         ldb     Bl, IOS1                           ; read HSO status register, status
399  0113 B0161C                                                       ;   flags are cleared upon read.
400
401                         jbc     Bl, 1, STI_10                      ; jump if Timer2 flag not set,
402  0116 311C03            call    Timer2                             ;    else Timer2 interrupt occured.
403  0119 EFFFFE
404                                                                    ; jump if Timer1 flag not set,
405  011C 301C03    STI_10: jbc     Bl, 0, STI_30                      ;    else Timer1 interrupt occured.
406  011F EFFFFE            call    Timer1
407
```

```
                                        ; other Timer interrupt routines to be processed
                                        ;   are placed here.

408
409  0122                STI_30:
410  0122 CC20                  pop     Dx
411  0124 CC1E                  pop     Cx                  ; restore pushed registers
412  0126 CC1C                  pop     Bx
413  0128 CC1A                  pop     Ax
414  012A F5                    popa
415  012B F0                    ret
416
417
```

```
;*****************************************************************************
;*                                                                           *
;*                              S E R I A L   I S R                          *
;*                                                                           *
;*****************************************************************************
;*  inputs      |   outputs    |   routines called   |   registers altered  *
;*****************************************************************************
;*  SP_STAT     |   INT_MASK   |   Rx_Int            |        None          *
;*                             |   Tx_Int            |                      *
;*                             |                     |                      *
;*****************************************************************************

;               Serial Port interrupt handler routine.

;


TI:     equ     5
RI:     equ     6

        org     Rom_Start + 200h

Serial_ISR:                                     ; push PSW, disable interrupts
        pushf
        push    Ax
        ldb     INT_MASK, #01000000b            ; re-enable serial interrupts
        ei

                                                ; read Serial Port Status
        ldb     Al, SP_STAT
        jbc     Al, TI, SI_1
        ljmp    Tx_int

SI_1:   jbc     Al, RI, SI_2
        ljmp    Rx_Int

                                                ; unknown interrupt
SI_2:   pop     Ax
        popf
        ret
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                                                                           *
                ;*                          E R R O R   I N T                                *
                ;*                                                                           *
                ;*****************************************************************************
                ;*  inputs     |   outputs    |   routines called   |   registers altered  *
                ;*****************************************************************************
                ;*  none       |                      none           |        none          *
                ;*****************************************************************************
                ;
                ;               This is the un-implimented interrupt error handling routine
                ;

021A            Error_Int:

                        push    Bx
021A C81C               push    Cx
021C C81E               push    Dx
021E C820               push    Fx
0220 C824                                                   ; Send detected interrupt number

                        call    Xmt_Char
0222 EFFFFE             ldb     Al, #' '
0225 B1201A             call    Xmt_Char
0228 EFFFFE                                                 ; send wild interrupt message

                        ld      Ax, #12
022B A10C001A           call    Send_Msg
022F EFFFFE

                        pop     Fx
0232 CC24               pop     Dx
0234 CC20               pop     Cx
0236 CC1E               pop     Bx
0238 CC1C               pop     Ax
023A CC1A               ret
023C F0

023D C81A       Error_Int_1:    push    Ax
023F B1311A                     ldb     Al, #'1'
0242 27D6                       sjmp    Error_Int

0244 C81A       Error_Int_2:    push    Ax
0246 B1321A                     ldb     Al, #'2'
0249 27CF                       sjmp    Error_Int

024B C81A       Error_Int_3:    push    Ax
024D B1331A                     ldb     Al, #'3'
0250 27C8                       sjmp    Error_Int

0252 C81A       Error_Int_4:    push    Ax
```

```
                                    ldb     Al, #'4'
0254 B1341A                         sjmp    Error_Int
0257 27C1
                 Error_Int_5:       push    Ax
0259 C81A                           ldb     Al, #'5'
025B B1351A                         sjmp    Error_Int
025E 27BA
                 Error_Int_6:       push    Ax
0260 C81A                           ldb     Al, #'6'
0262 B1361A                         sjmp    Error_Int
0265 27B3
                 Error_Int_7:       push    Ax
0267 C81A                           ldb     Al, #'7'
0269 B1371A                         sjmp    Error_Int
026C 27AC
                 Error_Int_8:       push    Ax
026E C81A                           ldb     Al, #'8'
0270 B1381A                         sjmp    Error_Int
0273 27A5
                 Error_Int_9:       push    Ax
0275 C81A                           ldb     Al, #'9'
0277 B1391A                         sjmp    Error_Int
027A 279E
                 Error_Int_A:       push    Ax
027C C81A                           ldb     Al, #'A'
027E B1411A                         sjmp    Error_Int
0281 2797
                 Error_Int_B:       push    Ax
0283 C81A                           ldb     Al, #'B'
0285 B1421A                         sjmp    Error_Int
0288 2790
                 Error_Int_C:       push    Ax
028A C81A                           ldb     Al, #'C'
028C B1431A                         sjmp    Error_Int
028F 2789
                 Error_Int_D:       push    Ax
0291 C81A                           ldb     Al, #'D'
0293 B1441A                         sjmp    Error_Int
0296 2782
                                    org     Rom_Start + 2FFh
02FF
                 Space:             DSB     1
02FF
```

```
557
558                     END
559  0300
                        #######
   Errors:  None        # ISR #
   Bytes:   249         #######
   CRC:     A2D3
```

SFM/ALP Interrupt Service Routines

Symbol and Cross Reference Table

Symbol	Value	Type	Defline	Refline		
Segment Definitions						
ISR	0000	S00R	16			
LOAD_ISR	0300	S01R	282			
External Symbols						
Asc_Hex	0000	X08	59			
Init_Serial	0000	X05	58			
Rcv_Char	0000	X00	58			
Rx_Int	0000	X03	58	456		
Send_Hex_Byte	0000	X07	59			
Send_Hex_Word	0000	X06	59			
Send_Msg	0000	X02	58	491		
Timer1	0000	X0B	63	406		
Timer2	0000	X0C	63	403		
Tx_int	0000	X04	58	453		
Wait_CR	0000	X09	59			
Wait_Tx_Inactive	0000	X0A	59			
Xmt_Char	0000	X01	58	486	488	

Public Symbols

Local Symbols

AD_COMMAND	0002	A‡	106	
AD_Complete	0020	R01	311	
AD_RESULT_HI	0003	A‡	105	
AD_RESULT_LO	0002	A‡	104	
Accel	0003	A	245	
Ah	001B	A‡	164	
Al	001A	A‡	165	451 452 455 487 501 505 509 513 517 521 525 529 533 537 541 545 549
Ax	001A	A‡	163	389 390 413 447 458 490 497 500 504 508 512 516 520 524 528 532 536 540 544 548
BAUD_RATE	000E	A‡	127	
Bel	0007	A	257	
Bl	001C	A‡	168	399 402 405
Bs	0008	A	255	
Bx	001C	A‡	166	393 412 481 496
CR_Detected	0005	A	214	
CS	0000	R01	285	
Ch	001F	A‡	170	

Cl	001E	A‡	171								
Cr	000D	A	253								
Cx	001E	A‡	169	394	411	482	495				
Day	000C	R01	290								
Del	007F	A	256								
Dh	0021	A‡	173								
Dl	0020	A‡	174								
Dx	0020	A‡	172	395	410	483	494				
Eh	0023	A‡	176								
El	0022	A‡	177								
Error_Int	021A	R01	479	502	506	510	514	518	522	526	
				530	534	538	542	546	550		
Error_Int_1	023D	R01	500	307							
Error_Int_2	0244	R01	504	311							
Error_Int_3	024B	R01	508	315							
Error_Int_4	0252	R01	512								
Error_Int_5	0259	R01	516	323							
Error_Int_6	0260	R01	520	327							
Error_Int_7	0267	R01	524	331							
Error_Int_8	026E	R01	528	335							
Error_Int_9											

0275 R01 532 339

Error_Int_A
027C R01 536 343

Error_Int_B
0283 R01 540 347

Error_Int_C
028A R01 544 351

Error_Int_D
0291 R01 548 355

Ex 0022 A‡ 175

ExInt_Pin
00B0 R01 347

FF 000C A 258

Fh 0025 A‡ 179

Fl 0024 A‡ 180

Flags 0031 A‡ 193

Flags2 0032 A‡ 194

Fx 0024 A‡ 178 484 493

HSI1 0050 R01 323

HSI4 0080 R01 335

HSI_Available
0030 R01 315

HSI_Full 00C0 R01 351

HSI_MODE 0003 A‡ 108

HSI_STATUS
0006 A‡ 110

HSI_TIME 0004 A‡ 109

HSO 0040 R01 319

```
O_C.      0006  A‡     113

O_TIME    0004  A‡     112

          0009  A      259

NT_MASK   0008  A‡     119     396   448

NT_MASK1  0013  A‡     157

NT_PEND   0009  A‡     120     392

NT_PEND1  0012  A‡     156

OC0       0015  A‡     139

OC1       0016  A‡     140

OC2       0017  A‡     160

IOS0      0015  A‡     136

IO.       0016  A‡     137     399

IOS2      0017  A‡     159

IO_PORT0  000E  A‡     129

IO_PORT1  000F  A‡     130

IO_PORT2  0010  A‡     131

IR_BASE   001A  A‡      84     86    87    89    90    91    92    93
                                94    95    96

Jiffy_Clock
          002E  A‡     190

Jiffy_Override
          0001  A      221

Key       0002  R01    286

Lf        000A  A      254

Month     000A  R01    289
```

NMI	00D0	R01	355							
OpCode	0070	R01	331							
PWM_CONTROL	0017	A‡	142							
Port1_init	0000	A	263							
Port2_init	00FF	A	264							
R0	001C	A‡	89							
R1	001D	A‡	90							
R2	001E	A‡	91							
R3	001F	A‡	92							
R4	0020	A‡	93							
R5	0021	A‡	94							
R6	0022	A‡	95							
R7	0023	A‡	96							
R8	001A	A‡	86							
R9	001B	A‡	87							
RI	0006	A	440	455						
Rcv_Empty	0004	A	213							
Rcv_Full	0003	A	212							
Rcv_In_Ptr	0026	A‡	185							
Rcv_Out_Ptr	0028	A‡	186							
Rom_Start	0000	R01	283	305	309	313	317	321	325	329

				333	337	341	345	349	353	384		
				442	552							
SBUF	0007	A‡	115									
SBUF_RX	0007	A‡	116									
SBUF_TX	0007	A‡	117									
SI_1	0210	R01	455	452								
SI_2	0216	R01	458	455								
SP	0018	A‡	101									
SP_CON	0011	A‡	134									
SP_STAT	0011	A‡	133	451								
STACKPOINTER	0018	A‡	102									
STI_10	011C	R01	405	402								
STI_30	0122	R01	409	405								
Serial_Flags	0030	A‡	192									
Serial_ISR	0200	R01	444									
Software_Timer_ISR	0100	R01	386	319								
Sp	0020	A	261									
Space	02FF	R01	554									
T2cap	0090	R01	339									
T2overflow	00A0	R01	343									
TI	0005	A	439	452								
TIMER1	000A	A‡	124	390								
TIMER2	000C	A‡	125									

T_Full	0004	A	246
T_Overflow	0010	R01	307
Timer1_Int	0000	A	220
Trap_Vector	0060	R01	327
Vt	000B	A	260
WATCHDOG	000A	A‡	122
WSR	0014	A‡	158
Xmt_Active	0000	A	209
Xmt_Empty	0002	A	211
Xmt_Full	0001	A	210
Xmt_In_Ptr	002A	A‡	187
Xmt_Out_Ptr	002C	A‡	188
Year	000E	R01	291
ZERO	0000	A‡	182
_Bh	001D	A‡	167
banking	0000	A‡	150

Macro Definitions

SFMALP.ASM

```
##########################################################################
#                                                                        #
#     Archimedes 8096/196 Assembler V1.95/MD2          13/Jul/92  04:03:30   #
#                                                                        #
#      Source   =   sfmalp.asm                                           #
#      List     =   sfmalp.lst                                           #
#      Object   =   sfmalp.obj                                           #
#      Options  =   s                                                    #
#                                                                        #
#                                 (c) Copyright Archimedes Software Inc. 1988 #
##########################################################################
```

```
                ;*************************************************************************
                ;*                                                                       *
                ;*                  S O L A R   F L U X   M O N I T O R                  *
                ;*                                                                       *
                ;*             A D V A N C E D   L A N G M U I R   P R O B E             *
                ;*                                                                       *
                ;*                          S F M / A L P                                *
                ;*                                                                       *
                ;*                                                                       *
                ;*                  Assembly Language Control Program                    *
                ;*                                                                       *
                ;*                           Version 0.9                                 *
                ;*                                                                       *
                ;*************************************************************************

                ;                     Assembler Directives
0000                    LSTCOD  +
0000                    LSTWID  -
0000                    LSTPAG  +
0000                    LSTXRF
0000                    PAGSIZ  52
```

29
30
31

```
                ;               ****************************************************
                ;               *                                                  *
                ;               *                                                  *
                ;               *                    P U B L I C S                 *
                ;               *                                                  *
                ;               *                                                  *
                ;               ****************************************************

                                ; Register Publics

0000            Public          Ax, Ah, Al, Bx, _Bh, Bl, Cx, Ch, Cl, Dx, Dh, Dl
0000            Public          Ex, Eh, El, Fx, Fh, Fl
0000            Public          Serial_Flags, Flags, Flags2, Status
0000            Public          SFM_Vern_PG, ALP_Vern_PG, SFM_VA_Val, ALP_VA_Val
0000            Public          Clock_ZP

                                ; External Address Publics

0000            Public          ZRAM_start, ZRAM_stop
0000            Public          EEPROM_start, EEPROM_stop, Loader_CS, Rom_CS

                                ; C callable routines
0000            Public          loader, delay, sleep, read_AD

                                ; ASM callable routines
0000            Public          Delay

                                ; IO Address Publics
0000            Public          Grid_Volt_WR, Timer_WR, VAGen_WR, EMeter_WR, EMeter_RD
0000            Public          Telem, Telem_WR, Probe_Cln_WR

                                ; Zero  Power Ram Publics

0000            Public          Rcv_Buffer, Xmt_Buffer, Rcv_End, Xmt_End
0000            Public          ALP_Mode_Ptr, ALP_Mode_Step, Curr_ALP_Mode, ALP_telem
0000            Public          SFM_Mode_Ptr, SFM_Mode_Step, Curr_SFM_Mode, SFM_telem
0000            Public          ALP_mode_start_time, SFM_mode_start_time
0000            Public          clean_word
0000            Public          HSKP_telem, IDLE_telem
```

```
                ;               ***************************************************
                ;               *                                                 *
                ;               *                                                 *
                ;               *                  E X T E R N A L S              *
                ;               *                                                 *
                ;               *                                                 *
                ;               ***************************************************

                ;
0000            Extern          main, ?SEG_INIT_L00

                ;               Serial Externals

0000            Extern  Rcv_Char, Xmt_Char, Send_Msg, Rx_Int, Tx_int, Init_Serial
0000            Extern  Send_Hex_Word, Send_Hex_Byte, Asc_Hex, Wait_CR, Wait_Tx_Inactive

                ;               Timer Externals

0000            Extern  Init_Timer1
```

```
 97
 98                ;
 99                ;               ***************************************************
100                ;               *                                                 *
101                ;               *      R E G I S T E R   D E F I N I T I O N S    *
102                ;               *                                                 *
103                ;               ***************************************************
104
105
106                $REG8096.INC                    ;INCLUDE FILE FOR 8096 REGISTERS
107                ;       - REG8096.INC -
108                ;
109                ;       Include file containing 8096 register definitions
110                ;
111
112                ;
113                ;       8096 C Compiler Working registers
114                ;
115  001A          IR_BASE DEFINE  01AH            ; Register base address
116
117  001A          R8      DEFINE  (IR_BASE)
118  001B          R9      DEFINE  (IR_BASE+1)
119
120  001C          R0      DEFINE  (IR_BASE+2)
121  001D          R1      DEFINE  (IR_BASE+3)
122  001E          R2      DEFINE  (IR_BASE+4)
123  001F          R3      DEFINE  (IR_BASE+5)
124  0020          R4      DEFINE  (IR_BASE+6)
125  0021          R5      DEFINE  (IR_BASE+7)
126  0022          R6      DEFINE  (IR_BASE+8)
127  0023          R7      DEFINE  (IR_BASE+9)
128
129                ;
130                ;       Special Function Registers
131                ;
132  0018          SP              DEFINE  18H
133  0018          STACKPOINTER    DEFINE  18H
134
135  0002          AD_RESULT_LO    DEFINE  02H
136  0003          AD_RESULT_HI    DEFINE  03H
137  0002          AD_COMMAND      DEFINE  02H
138
139  0003          HSI_MODE        DEFINE  03H
140  0004          HSI_TIME        DEFINE  04H
141  0006          HSI_STATUS      DEFINE  06H
142
143  0004          HSO_TIME        DEFINE  04H
144  0006          HSO_COMMAND     DEFINE  06H
145
```

```
0007        SBUF          DEFINE  07H
0007        SBUF_RX       DEFINE  07H
0007        SBUF_TX       DEFINE  07H

0008        INT_MASK      DEFINE  08H
0009        INT_PEND      DEFINE  09H

000A        WATCHDOG      DEFINE  0AH

000A        TIMER1        DEFINE  0AH
000C        TIMER2        DEFINE  0CH

000E        BAUD_RATE     DEFINE  0EH

000E        IO_PORT0      DEFINE  0EH
000F        IO_PORT1      DEFINE  0FH
0010        IO_PORT2      DEFINE  10H

0011        SP_STAT       DEFINE  11H
0011        SP_CON        DEFINE  11H

0015        IOS0          DEFINE  15H
0016        IOS1          DEFINE  16H

0015        IOC0          DEFINE  15H
0016        IOC1          DEFINE  16H

0017        PWM_CONTROL   DEFINE  17H

            $DEF8096.INC
            ;       - DEF8096.INC -
            ;
            ;       Include file for getting a nonbanked model
            ;

0000        banking       DEFINE  0



            $SFMALP.INC
                                          ; 80C196 Special Function Registers
0012        INT_PEND1     DEFINE          12h
0013        INT_MASK1     DEFINE          13h
0014        WSR           DEFINE          14h
0017        IOS2          DEFINE          17h
0017        IOC2          DEFINE          17h

                                          ; Registers to be used in assembly routines
001A        Ax            DEFINE          1Ah
```

```
001B    Ah          DEFINE      1Bh
001A    Al          DEFINE      1Ah
001C    Bx          DEFINE      1Ch
001D    _Bh         DEFINE      1Dh
001C    Bl          DEFINE      1Ch
001E    Cx          DEFINE      1Eh
001F    Ch          DEFINE      1Fh
001E    Cl          DEFINE      1Eh
0020    Dx          DEFINE      20h
0021    Dh          DEFINE      21h
0020    Dl          DEFINE      20h
0022    Ex          DEFINE      22h
0023    Eh          DEFINE      23h
0022    El          DEFINE      22h
0024    Fx          DEFINE      24h
0025    Fh          DEFINE      25h
0024    Fl          DEFINE      24h

0000    ZERO        DEFINE      00h ; To be used in comparisons

                                ; Serial Port, Jiffy Clock, and Flags
0026    Rcv_In_Ptr  DEFINE      26h
0028    Rcv_Out_Ptr DEFINE      28h
002A    Xmt_In_Ptr  DEFINE      2Ah
002C    Xmt_Out_Ptr DEFINE      2Ch

002E    Jiffy_Clock DEFINE      2Eh

0030    Serial_Flags DEFINE     30h
0031    Flags       DEFINE      31h
0032    Flags2      DEFINE      32h
```

```
005D EF0001    RI_11:  call    Set_Rcv_Empty          ; cannot delete from empty buffer.
0060 CC1A              pop     Ax                     ; exit ISR.
0062 F3                popf
0063 F0                ret

0064 0526      RI_12:  dec     Rcv_In_Ptr             ; remove deleted character from Rcv Buff
0066 89000026          cmp     Rcv_In_Ptr, #Rcv_Buffer ; check for buffer under flow
006A D604              jge     RI_13                  ; If underflow. set ptr to end of buffer
006C A1FFFF26          ld      Rcv_In_Ptr, #(Rcv_End - 1)
0070 EFE900    RI_13:  call    Clr_Rcv_Full           ; Clear receive full flag. char deleted.
0073 882826            cmp     Rcv_In_Ptr, Rcv_Out_Ptr ; Check for empty buffer.
0076 D703              jne     RI_14
0078 EFE500            call    Set_Rcv_Empty          ; If empty set empty flag.
007B CC1A      RI_14:  pop     Ax                     ; exit from receive ISR.
007D F3                popf
007E F0                ret
```

```
007F 990A1B    RI_20:  cmpb    Ah, #Lf                  ; check for received line feed.
0082 D704              jne     RI_30                    ; ignore line feeds.
0084 CC1A              pop     Ax
0086 F3                popf
0087 F0                ret
                                                        ; don't stuff char if buffer is full
0088 3B300D    RI_30:  jbs     Serial_Flags, Rcv_Full, RI_40
008B C6271B            stb     Ah, [Rcv_In_Ptr]+        ; store character into buffer
008E 89000026          cmp     Rcv_In_Ptr, #Rcv_End     ; check for buffer overflow
0092 DE04              jlt     RI_40
0094 A1000026          ld      Rcv_In_Ptr, #Rcv_Buffer ; if overflow. reset to top of buffer

0098 B01B1A    RI_40:  ldb     Al, Ah                   ; Echo character
009B EF7D00            call    Xmt_Char

009E 990D1B            cmpb    Ah, #Cr                  ; check for received carriage return
00A1 D709              jne     RI_50
00A3 EFC200            call    Set_CR_Detected          ; if set set Cr flag and send line feed
00A6 B10A1A            ldb     Al, #Lf
00A9 EF6F00            call    Xmt_Char

00AC EFB500    RI_50:  call    Clr_Rcv_Empty            ; clear receive empty flag
00AF 882826            cmp     Rcv_In_Ptr, Rcv_Out_Ptr ; check for full buffer. In ptr = Out
00B2 D703              jne     RI_60
00B4 EFA100            call    Set_Rcv_Full             ; If full, set flag.
00B7 CC1A      RI_60:  pop     Ax                       ; exit receive ISR
00B9 F3                popf
00BA F0                ret
```

```
418
419                 ;*****************************************************************************
420                 ;*                                                                           *
421                 ;*                                                                           *
422                 ;*                                T X    I N T                               *
423                 ;*                                                                           *
424                 ;*                                                                           *
425                 ;*****************************************************************************
426                 ;* inputs      |    outputs    |    routines called    |   registers altered *
427                 ;*****************************************************************************
428                 ;*   SP_STAT    |   INT_MASK    |     Rx_Int            |       None          *
429                 ;*              |               |     Tx_Int            |                     *
430                 ;*              |               |                       |                     *
431                 ;*              |               |                       |                     *
432                 ;*****************************************************************************
433
434
435  00BB           Tx_int:                 ; Handle Transmit interrupt
436
437  00BB 383004            jbs     Serial_Flags, Xmt_Active, TI_10
438  00BE CC1A              pop     Ax                      ; Abort if transmitter is not active
439  00C0 F3                popf
440  00C1 F0                ret
441                                                         ; If buffer full, don't ck if empty
442  00C2 393012    TI_10:  jbs     Serial_Flags, Xmt_Full, TI_20
443  00C5 882C2A            cmp     Xmt_In_Ptr, Xmt_Out_Ptr ; ck if (in ptr = out ptr) and not full
444  00C8 D70D              jne     TI_20                   ;    then buffer = empty.
445  00CA EF8300            call    Set_Xmt_Empty           ; set serial flags and exit interrupt
446  00CD EF7C00            call    Clr_Xmt_Full
447  00D0 EF7100            call    Clr_Xmt_Active
448  00D3 CC1A              pop     Ax
449  00D5 F3                popf
450  00D6 F0                ret
451
452  00D7           Send:
453  00D7 B22D07    TI_20:  ldb     SBUF, [Xmt_Out_Ptr]+    ; Transmit character from xmt buffer
454  00DA 8900002C          cmp     Xmt_Out_Ptr, #Xmt_End   ; Ck for buffer wrap around
455  00DE DE04              jlt     TI_30
456  00E0 A100002C          ld      Xmt_Out_Ptr, #Xmt_Buffer
457  00E4 EF5900    TI_30:  call    Set_Xmt_Active          ; set serial flags and exit
458  00E7 EF6200            call    Clr_Xmt_Full
459  00EA CC1A              pop     Ax
460  00EC F3                popf
461  00ED F0                ret
462
463
```

```
465
466                 ;*****************************************************************************
467                 ;*                                                                           *
468                 ;*                                                                           *
469                 ;*                          R C V     C H A R                                *
470                 ;*                                                                           *
471                 ;*                                                                           *
472                 ;*****************************************************************************
473                 ;* inputs      |    outputs    |    routines called    |   registers altered *
474                 ;*****************************************************************************
475                 ;* Rcv_Buffer  |   Ser_Flags   |     Set_Rcv_Empty     |     Rcv_Out_Ptr     *
476                 ;*             |      A1       |     Clr_Rcv_Full      |                     *
477                 ;*             |               |    Clr_CR_Detected    |                     *
478                 ;*****************************************************************************
479
480                 ;               This routine pulls a character out of the receive buffer
481                 ;       and returns it in A1.  If the Buffer is empty a null ( 0h )
482                 ;       character is returned in A1.
483                 ;               If a carriage return is read then the CR_detect flag is
484                 ;       cleared. If the buffer is emptied then the Rcv_Empty flag is set.
485                 ;       Rcv_Full is always cleared.
486                 ;               If a null is then the CR_Detected flag is always cleared.
487
488
489  00EE           Rcv_Char:
490
491  00EE F2                pushf                          ; save flags, disable interrupts
492  00EF EF6A00            call    Clr_Rcv_Full           ; clear receive full flag.
493                                                        ; check if receive buffer is empty.
494  00F2 343007            jbc     Serial_Flags, Rcv_Empty, RC_10
495  00F5 EF7400            call    Clr_CR_Detected        ; clear Cr detected flag
496  00F8 111A              clrb    A1                     ; if empty return a zero in A1.
497  00FA F3                popf                           ; restore flags & interrupts
498  00FB F0                ret
499
500  00FC B2291A    RC_10:  ldb     A1, [Rcv_Out_Ptr]+     ; read character from receive buffer.
501  00FF 990D1A            cmpb    A1, #Cr                ; check if character is a carriage rtn.
502  0102 D703              jne     RC_20
503  0104 EF6500            call    Clr_CR_Detected        ; clear CR flag.
504
505  0107 89000028  RC_20:  cmp     Rcv_Out_Ptr, #Rcv_End  ; check for buffer overflow.
506  010B D704              jne     RC_30                  ; If overflowed,
507  010D A1000028          ld      Rcv_Out_Ptr, #Rcv_Buffer;   set pointer to top of buffer.
508
509  0111 882628    RC_30:  cmp     Rcv_Out_Ptr, Rcv_In_Ptr ; check if buffer empty.
510  0114 D703              jne     RC_40                  ; If Empty,
511  0116 EF4700            call    Set_Rcv_Empty          ;    set empty flag.
512
513  0119 F3        RC_40:  popf                           ; restore flags & interrupts
514  011A F0                ret
515
516
```

```
518
519             ;*****************************************************************************
520             ;*                                                                           *
521             ;*                                                                           *
522             ;*                          X M T     C H A R                                *
523             ;*                                                                           *
524             ;*                                                                           *
525             ;*****************************************************************************
526             ;* inputs      |    outputs    |    routines called    |   registers altered *
527             ;*****************************************************************************
528             ;*    Al       |   Ser_Flags   |        none           |     Xmt_In_Ptr      *
529             ;*             |   Xmt_Buffer  |                       |                     *
530             ;*****************************************************************************
531
532             ;               This routine stuff the character in variable Al into
533             ;       the transmit buffer.  If the transmitter is not currently active
534             ;       then it is activated and the character is transmitted.
535             ;               If In Pointer = Out Pointer then the transmitt buffer is
536             ;       full and this routine will wait until it emptys before stuffing
537             ;       the data in CHAR into the buffer.
538
539
540  011B       Xmt_Char:       Equ     $
541
542  011B 3930FD            jbs     Serial_Flags, Xmt_Full, Xmt_Char; If Xmt Buff full, Wait until
543                                                             ;    it emptys.
544
545  011E F2                pushf                               ; save flags, disable intrrupts.
546  011F C62B1A            stb     Al, [Xmt_In_Ptr]+           ; Store Character into Buffer,
547                                                             ;    bump buffer pointer
548  0122 EF2F00            call    Clr_Xmt_Empty               ; Clear empty flag
549  0125 8900002A          cmp     Xmt_In_Ptr, # Xmt_End       ; Ck for buffer overflow
550  0129 DE04              jlt     XC_10                       ; If Pointer < End then jump,
551  012B A100002A          ld      Xmt_In_Ptr, # Xmt_Buffer    ;   else set ptr to buffer top
552
553  012F 882C2A    XC_10:  cmp     Xmt_In_Ptr, Xmt_Out_Ptr     ; check for In_Ptr = Out_Ptr
554  0132 D703              jne     XC_20                       ;     If true then
555  0134 EF1100            call    Set_Xmt_Full                ;     Xmt Buffer is full
556
557  0137 383004    XC_20:  jbs     Serial_Flags, Xmt_Active, XC_30 ; jump if transmitter is active
558  013A C81A              push    Ax                          ;   enter Xmt Intr ISR.
559  013C 2799              sjmp    Send                        ;   and activate it.
560
561  013E F3        XC_30:  popf                                ; restrore flags & interrupts
562  013F F0                ret
563
564
565
566
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                S E R I A L   F L A G   U T I L I T I E S                  *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs     |    routines called    |   registers altered *
                ;*****************************************************************************
                ;*    none     |   Ser_Flags    |        none           |         none        *
                ;*****************************************************************************

0140            Set_Xmt_Active:
0140 910130             orb     Serial_Flags, #00000001b
0143 F0                 ret

0144            Clr_Xmt_Active:
0144 71FE30             andb    Serial_Flags, #11111110b
0147 F0                 ret

0148            Set_Xmt_Full:
0148 910230             orb     Serial_Flags, #00000010b
014B F0                 ret

014C            Clr_Xmt_Full:
014C 71FD30             andb    Serial_Flags, #11111101b
014F F0                 ret

0150            Set_Xmt_Empty:
0150 910430             orb     Serial_Flags, #00000100b
0153 F0                 ret

0154            Clr_Xmt_Empty:
0154 71FB30             andb    Serial_Flags, #11111011b
0157 F0                 ret

0158            Set_Rcv_Full:
0158 910830             orb     Serial_Flags, #00001000b
015B F0                 ret

015C            Clr_Rcv_Full:
015C 71F730             andb    Serial_Flags, #11110111b
015F F0                 ret

0160            Set_Rcv_Empty:
0160 911030             orb     Serial_Flags, #00010000b
0163 F0                 ret

0164            Clr_Rcv_Empty:
0164 71EF30             andb    Serial_Flags, #11101111b
0167 F0                 ret
```

```
0168            Set_CR_Detected:
0168 912030             orb     Serial_Flags, #00100000b
016B F0                 ret

016C            Clr_CR_Detected:
016C 71DF30             andb    Serial_Flags, #11011111b
016F F0                 ret
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                                                                           *
                ;*                          S E N D    M S G                                 *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs    |    routines called    |   registers altered *
                ;*****************************************************************************
                ;*    Ax       |    Xmt_Buff   |       Xmt_Char        |         none        *
                ;*****************************************************************************

                ;               This routine scans the message table for the message number
                ;       contained in Ax.  It does this by counting the zeros that terminate
                ;       each message in the table.  Once a message has been found it is
                ;       stuffed into the transmitt buffer and sent out.

 0170           Send_Msg:
 0170 88001A            cmp     Ax, ZERO                ; check if message number = 0
 0173 D701              jne     SM_10                   ;    if it does then abort.
 0175 F0                ret

 0176 C81C      SM_10:  push    Bx
 0178 C81E              push    Cx
 017A A101001E          ld      Cx, #1                  ; initialize Cx to message number 1
 017E A104021C          ld      Bx, # Message_Table     ; initialize Bx as table pointer

 0182 881E1A    SM_20:  cmp     Ax, Cx                  ; check if message has been found.
 0185 DF0F              je      SM_40                   ;     jump if found
 0187 89D3021C  SM_30:  cmp     Bx, #Msg_Table_End      ; abort upon end of message table
 018B DF15              je      SM_50
 018D 9A1D00            cmpb    ZERO, [Bx]+             ; search for next zero in table
 0190 D7F5              jne     SM_30
 0192 071E              inc     Cx                      ; bump message found counter
 0194 27EC              sjmp    SM_20

 0196 B21D1A    SM_40:  ldb     Al, [Bx]+               ; read message character from table
 0199 981A00            cmpb    ZERO, Al                        ; ck for end of message zero
 019C DF04              je      SM_50
 019E 2F7B              call    Xmt_Char                ; stuff table character into Xmt Buffer
 01A0 27F4              sjmp    SM_40                   ; read next character from table.

 01A2 CC1E      SM_50:  pop     Cx                      ; restore registers
 01A4 CC1C              pop     Bx
 01A6 F0                ret
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                          S E N D   H E X   W O R D                        *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs    |    routines called    |   registers altered *
                ;*****************************************************************************
                ;*     Ax      |      none     |      Send_Hex_Byte    |        none         *
                ;*****************************************************************************

                ;               This routine transmits the word passed in Ax as a 16 bit
                ;       ascii encoded hex

01A7            Send_Hex_Word:

01A7 C81A               push    Ax

01A9 08081A             shr     Ax, #8                      ; xfer upper byte into Al
01AC EF0600             call    Send_Hex_Byte               ; send upper byte of word
01AF CC1A               pop     Ax                          ; recall word
01B1 EF0100             call    Send_Hex_Byte               ; send lower byte

01B4 F0                 ret
```

```
712                 ;*****************************************************************************
713                 ;*                                                                           *
714                 ;*                                                                           *
715                 ;*                          S E N D   B Y T E   H E X                        *
716                 ;*                                                                           *
717                 ;*                                                                           *
718                 ;*****************************************************************************
719                 ;* inputs      |    outputs    |    routines called    |    registers altered *
720                 ;*****************************************************************************
721                 ;*     Al      |      none     |        Xmt_Char       |        none         *
722                 ;*****************************************************************************
723
724                 ;               This routine transmits the byte passed in Al as an 8 bit
725                 ;       ascii encoded hex.
726
727
728
729  01B5           Send_Hex_Byte:
730
731  01B5 C81A              push    Ax                      ; shift upper nibble into lower.
732  01B7 18041A            shrb    Al, #4
733  01BA EF0D00            call    SBH_10
734  01BD CC1A              pop     Ax
735  01BF C81A              push    Ax
736  01C1 710F1A            andb    Al, #00001111b          ; mask off upper byte
737  01C4 EF0300            call    SBH_10
738  01C7 CC1A              pop     Ax
739  01C9 F0                ret
740
741
742  01CA 75301A    SBH_10: addb    Al, #30h                ; convert Al to Ascii
743  01CD 993A1A            cmpb    Al, #3Ah                ; ck if result > ascii (9)
744  01D0 DE03              jlt     SBH_20
745  01D2 75071A            addb    Al, #7                  ; finish conversion
746  01D5 2F44      SBH_20: call    Xmt_Char
747  01D7 F0                ret
748
```

```
750                 ;*****************************************************************
751                 ;*                                                               *
752                 ;*                                                               *
753                 ;*                          A S C    H E X                       *
754                 ;*                                                               *
755                 ;*                                                               *
756                 ;*****************************************************************
757                 ;* inputs     |   outputs    |   routines called   |  registers altered *
758                 ;*****************************************************************
759                 ;*     Al     |     Al       |        none         |       none         *
760                 ;*            |     C        |                     |                    *
761                 ;*****************************************************************
762
763                 ;           This routine converts the character from ASCII to Hex.  If the
764                 ;      conversion compleates ok then the carry flag is cleared and the result
765                 ;      is returned in Al.  If the character is not 0-F then carry is set and
766                 ;      a zero is returned in Al.
767
768
769  01D8           Asc_Hex:
770  01D8 79301A            subb    Al, #'0'                ; subtract ASCII '0'
771  01DB D311              jnc     AH_10                   ; non ASCII on borrow. C = /borrow.
772  01DD 990A1A            cmpb    Al, #10
773  01E0 DE0A              jlt     AH_20
774  01E2 79071A            subb    Al, #7                  ; jump on borrow. Non ASCII
775  01E5 D307              jnc     AH_10
776  01E7 990F1A            cmpb    Al, #0Fh
777  01EA D202              jgt     AH_10
778  01EC F8        AH_20:  clrc                            ; clear carry.
779  01ED F0                ret
780                                                         ; set carry on non ASCII
781  01EE F9        AH_10:  setc
782  01EF 111A              clrb    Al
783  01F1 F0                ret
784
785
786
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                                                                           *
                ;*                              W A I T   C R                                *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs    |    routines called    |    registers altered *
                ;*****************************************************************************
                ;*    none     |      none     |        none           |        none          *
                ;*             |               |                       |                      *
                ;*****************************************************************************

                Wait_CR:

01F2

                        jbc     Serial_Flags, CR_Detected, Wait_CR
01F2 3530FD
01F5 F0                 ret
```

```
809
810             ;*****************************************************************************
811             ;*                                                                           *
812             ;*                                                                           *
813             ;*                          W A I T   T X   I N A C T I V E                  *
814             ;*                                                                           *
815             ;*                                                                           *
816             ;*****************************************************************************
817             ;* inputs      |   outputs    |   routines called   |   registers altered  *
818             ;*****************************************************************************
819             ;*    none     |     none     |        none         |        none          *
820             ;*             |              |                     |                      *
821             ;*****************************************************************************
822
823             ;               Waits until the serial transmitter is inactive.
824
825  01F6       Wait_Tx_Inactive:
826
827  01F6 3030FD        jbc     Serial_Flags, Xmt_Active, Wait_Tx_Inactive
828  01F9 F0            ret
829
830
831
```

```
                ;*************************************************************************
                ;*                                                                       *
                ;*                                                                       *
                ;*                                                                       *
                ;*                          C L E A R   C O M                            *
                ;*                                                                       *
                ;*                                                                       *
                ;*************************************************************************
                ;*  inputs     |   outputs    |   routines called   |   registers altered  *
                ;*************************************************************************
                ;*     none    |      none    |        none         |        none          *
                ;*             |              |                     |                      *
                ;*************************************************************************


01FA            Clear_Com:

                        ld      Rcv_Out_Ptr, Rcv_In_Ptr
01FA A02628             call    Set_Rcv_Empty
01FD 2F61               call    Clr_Rcv_Full
01FF 2F5B               call    Clr_CR_Detected
0201 2F69               ret
0203 F0
```

```
857                 ;*****************************************************************************
858                 ;*                                                                           *
859                 ;*                                                                           *
860                 ;*                          M E S S A G E   T A B L E                        *
861                 ;*                                                                           *
862                 ;*                                                                           *
863                 ;*****************************************************************************
864                 ;* inputs      |    outputs    |    routines called    |   registers altered *
865                 ;*****************************************************************************
866                 ;*    none     |      none     |          none         |         none        *
867                 ;*****************************************************************************
868
869                 ;                Look up table for serial port messages
870
871
872  0204           Message_Table:
873                                                                                  ; Msg #1
874  0204 070D0A0A          dcb     Bel, Cr, Lf, Lf, Lf
     0208 0A                                                  , 00
875  0209 00                dcb     ''                        , 00                    ; Msg #2
876  020A 00                dcb     ''                                                ; Msg #3
877  020B                   dcb     ''                        , 00
878  020B 00                dcb     'ZRAM cksum Error', Bel,Cr,Lf   , 00              ; Msg #4
879  020C 5A52414D
     0210 20636B73
     0214 756D2045
     0218 72726F72
     021C 070D0A00
880  0220 5A52414D          dcb     'ZRAM test FAILED', Bel,Cr,Lf   , 00              ; Msg #5
     0224 20746573
     0228 74204641
     022C 494C4544
     0230 070D0A00
881  0234 524F4D20          dcb     'ROM cksum Error' , Bel,Cr,Lf   , 00              ; Msg #6
     0238 636B7375
     023C 6D204572
     0240 726F7207
     0244 0D0A00
882  0247 00                dcb     ''                        , 00                    ; Msg #7
883  0248 5A52414D          dcb     'ZRAM ok'         , Cr,Lf       , 00              ; Msg #8
     024C 206F6B0D
     0250 0A00
884  0252 524F4D20          dcb     'ROM ok'          , Cr,Lf       , 00              ; Msg #9
     0256 6F6B0D0A
     025A 00                                                  , 00                    ; Msg #10
885  025B 00                dcb     ''                        , 00                    ; Mag #11
886  025C 00                dcb     ''                        , 00                    ; Msg #12
887  025D 00                dcb     ''                        , 00                    ; Msg #13
888  025E 00                dcb     ''                        , 00                    ; Msg #14
889  025F 00                dcb     ''                        , 00                    ; Msg #15
890  0260 54656C65          dcb     'Telemetry Buffer Full',Bel,Cr,Lf,00
     0264 6D657472
     0268 79204275
```

```
     026C 66666572
     0270 2046756C
     0274 6C070D0A
     0278 00
891  0279 54656C65        dcb    'Telemetry Buffer OK',Cr,Lf     , 00          ; Msg #16
     027D 6D657472
     0281 79204275
     0285 66666572
     0289 204F4B0D
     028D 0A00
892  028F 54656C65        dcb    'Telemetry FAILED', Bel,Cr,Lf   , 00          ; Msg #17
     0293 6D657472
     0297 79204641
     029B 494C4544
     029F 070D0A00
893  02A3 43616E6E        dcb    'Cannot fill TM Buff',Bel,Cr,Lf , 00          ; Msg #18
     02A7 6F742066
     02AB 696C6C20
     02AF 544D2042
     02B3 75666607
     02B7 0D0A00                                                               ; Msg #19
894  02BA 00              dcb    ''                              , 00          ; Msg #20
895  02BB 00              dcb    ''                              , 00          ; Msg #21
896  02BC 524F4D20        dcb    'ROM CS <> Loader CS', Bel,Cr,Lf, 00
     02C0 4353203C
     02C4 3E204C6F
     02C8 61646572
     02CC 20435307
     02D0 0D0A00
897
898  02D3            Msg_Table_End:
899
900  02D3            END

     Errors:  None        #######
     Bytes:   723         # Ser #
     CRC:     C346        #######
```

Symbol and Cross Reference Table

Symbol	Value	Type	Defline	Refline				
Segment Definitions								
Ser	02D3	S00R	17					
External Symbols								
Rcv_Buffer	0000	X00	238	298	299	374	398	507
Rcv_End	0000	X02	238	376	396	505		
Xmt_Buffer	0000	X01	238	301	302	456	551	
Xmt_End	0000	X03	238	454	549			
Public Symbols								
Asc_Hex	01D8	R00E	769	242				
Clear_Com	01FA	R00E	848	243				
Init_Serial	0000	R00E	289	240				
Rcv_Char	00EE	R00E	489	241				
Rx_Int	0021	R00E	335	240				
Send_Hex_Byte	01B5	R00E	729	242	703	705		
Send_Hex_Word	01A7	R00E	698	242				
Send_Msg	0170	R00E	649	241				
Tx_int	00BB	R00E	435	240				
Wait_CR	01F2	R00E	802	242	804			
Wait_Tx_Inactive								

```
         01F6  R00E   825     241   827

Xmt_Char 011B  R00E   540     241   356   358   360   401   407   542
                             671   746


                    Local Symbols
                    =============

AD_COMMAND
         0002  A#      63

AD_RESULT_HI
         0003  A#      62

AD_RESULT_LO
         0002  A#      61

AH_10    01EE  R00     781     771   775   777

AH_20    01EC  R00     778     773

Accel    0003  A       202

Ah       001B  A#      121     337   339   343   350   352   387   395
                             400   403

Al       001A  A#      122     355   357   359   400   406   496   500
                             501   546   668   669   732   736   742
                             743   745   770   772   774   776   782

Ax       001A  A#      120     346   369   381   389   413   438   448
                             459   558   650   659   700   702   704
                             731   734   735   738

BAUD_RATE
         000E  A#       84     292   293

Bel      0007  A       214     874   879   880   881   890   892   893
                             896

Bl       001C  A#      125

Bs       0008  A       212     352   355   359

Bx       001C  A#      123     654   657   661   663   668   675

CR_Detected
         0005  A       171     804

Ch       001F  A#      127

Cl       001E  A#      128
```

Clr_CR_Detected	016C	R00	625	495	503	853					
Clr_Rcv_Empty	0164	R00	617	409							
Clr_Rcv_Full	015C	R00	609	377	492	852					
Clr_Xmt_Active	0144	R00	585	447							
Clr_Xmt_Empty	0154	R00	601	548							
Clr_Xmt_Full	014C	R00	593	446	458						
Cr	000D	A	210	403	501	874	879	880	881	883	
				884	890	891	892	893	896		
Cx	001E	A‡	126	655	656	659	665	674			
Del	007F	A	213	350							
Dh	0021	A‡	130								
Dl	0020	A‡	131								
Dx	0020	A‡	129								
Eh	0023	A‡	133								
El	0022	A‡	134								
Ex	0022	A‡	132								
FF	000C	A	215								
Fh	0025	A‡	136								
Fl	0024	A‡	137								
Flags	0031	A‡	150	345							
Flags2	0032	A‡	151								
Fx	0024	A‡	135								
HSI_MODE	0003	A‡	65								
HSI_STATUS											

	0006	A‡	67								
HSI_TIME	0004	A‡	66								
HSO_COMMAND	0006	A‡	70								
HSO_TIME	0004	A‡	69								
Ht	0009	A	216								
INT_MASK	0008	A‡	76	309							
INT_MASK1	0013	A‡	114								
INT_PEND	0009	A‡	77								
INT_PEND1	0012	A‡	113								
IOC0	0015	A‡	96								
IOC1	0016	A‡	97								
IOC2	0017	A‡	117								
IOS0	0015	A‡	93								
IOS1	0016	A‡	94								
IOS2	0017	A‡	116								
IO_PORT0	000E	A‡	86								
IO_PORT1	000F	A‡	87								
IO_PORT2	0010	A‡	88								
IR_BASE	001A	A‡	41	43	44	46	47	48	49	50	
				51	52	53					
Jiffy_Clock	002E	A‡	147								
Jiffy_Override	0001	A	178								
Lf	000A	A	211	387	406	874	879	880	881	883	
				884	890	891	892	893	896		
Message_Table	0204	R00	872	657							

Msg_Table_End	02D3	R00	898	661	
PWM_CONTROL	0017	A‡	99		
Port1_init	0000	A	220		
Port2_init	00FF	A	221		
R0	001C	A‡	46		
R1	001D	A‡	47		
R2	001E	A‡	48		
R3	001F	A‡	49		
R4	0020	A‡	50		
R5	0021	A‡	51		
R6	0022	A‡	52		
R7	0023	A‡	53		
R8	001A	A‡	43		
R9	001B	A‡	44		
RC_10	00FC	R00	500	494	
RC_20	0107	R00	505	502	
RC_30	0111	R00	509	506	
RC_40	0119	R00	513	510	
RI_00	002A	R00	343	340	
RI_01	0036	R00	350	344	
RI_10	0040	R00	355	351	
RI_11	005D	R00	368	362	
RI_12	0064	R00	373	364	366
RI_13	0070	R00	377	375	

RI_14	007B	R00	381	379							
RI_20	007F	R00	387	353							
RI_30	0088	R00	394	388							
RI_40	0098	R00	400	394	397						
RI_50	00AC	R00	409	404							
RI_60	00B7	R00	413	411							
Rcv_Empty	0004	A	170	362	494						
Rcv_Full	0003	A	169	364	394						
Rcv_In_Ptr	0026	A‡	142	298	365	373	374	376	378	395	
				396	398	410	509	850			
Rcv_Out_Ptr	0028	A‡	143	299	365	378	410	500	505	507	
				509	850						
SBH_10	01CA	R00	742	733	737						
SBH_20	01D5	R00	746	744							
SBUF	0007	A‡	72	337	453						
SBUF_RX	0007	A‡	73								
SBUF_TX	0007	A‡	74								
SM_10	0176	R00	654	651							
SM_20	0182	R00	659	666							
SM_30	0187	R00	661	664							
SM_40	0196	R00	668	660	672						
SM_50	01A2	R00	674	662	670						
SP	0018	A‡	58								
SP_CON	0011	A‡	91	295							
SP_STAT	0011	A‡	90								
STACKPOINTER	0018	A‡	59								

Send	00D7	R00	452	559								
Serial_Flags	0030	A‡	149	305	362	364	394	437	442	494		
				542	557	582	586	590	594	598		
				602	606	610	614	618	622	626		
				804	827							
Set_CR_Detected	0168	R00	621	405								
Set_Rcv_Empty	0160	R00	613	368	380	511	851					
Set_Rcv_Full	0158	R00	605	412								
Set_Xmt_Active	0140	R00	581	457								
Set_Xmt_Empty	0150	R00	597	445								
Set_Xmt_Full	0148	R00	589	555								
Sp	0020	A	218									
TIMER1	000A	A‡	81									
TIMER2	000C	A‡	82									
TI_10	00C2	R00	442	437								
TI_20	00D7	R00	453	442	444							
TI_30	00E4	R00	457	455								
T_Full	0004	A	203									
Timer1_Int	0000	A	177									
Vt	000B	A	217									
WATCHDOG	000A	A‡	79									
WSR	0014	A‡	115									
XC_10	012F	R00	553	550								
XC_20	0137	R00	557	554								

XC_30	013E	R00	561	557					
Xmt_Active	0000	A	166	437	557	827			
Xmt_Empty	0002	A	168						
Xmt_Full	0001	A	167	442	542				
Xmt_In_Ptr	002A	A‡	144	301	443	546	549	551	553
Xmt_Out_Ptr	002C	A‡	145	302	443	453	454	456	553
ZERO	0000	A‡	139	650	663	669			
_Bh	001D	A‡	124						
banking	0000	A‡	107						

Macro Definitions

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    13/Jul/92  04:04:59

        Target option =  80C196
        Memory model  =  large
        Source file   =  intwri.c
        List file     =  intwri.lst
        Object file   =  intwri.rl5
        Command line  =  intwri.c -ml -vl -z -P -e -L -F -p52
                                                  (c) Copyright Archimedes Software Inc. 1990
```

- INTWRI.C -

```
1           /*
2
3             Reduced version of the "printf" function.
4
5             This module can be adopted to user-defined routines that needs formatters
6             with special properties like different output channels or new format
7             specifiers.
8
9             This reduced version of formatter is suitable when program size is critical
10            rather than formatting power.  This routine uses less than 20 bytes of
11            stack space which makes it practical even in systems with less than 256
12            bytes of user RAM.
13
14            The only formatting specifiers supported are:
15
16                    %%  %o  %d  %c  %s  %x  %l
17
18            It means that real variables are not supported as well as field
19            width and precision arguments.  Return value is always zero.
20
21            Version: 3.00 [IANR]
22
23            *** ADDED %l specifier 2-12-91  J.D. ***
24
25          */
26
27          #include <stdio.h>
28          #include <stdarg.h>
29
30          extern  void    Send_Hex_Word (int);
31
32
33
34          int printf (const char *format, ...)
35            {
36              static const char hex[] = "0123456789ABCDEF";
```

```
37          char format_flag;
38          unsigned int u_val, div_val, base;
39          unsigned long lu_val, ldiv_val;
40          int   temp;
41          char *ptr;
42          va_list ap;
43
44          va_start (ap, format);
45
46          for (;;)    /* Until full format string read */
47            {
48              while ((format_flag = *format++) != '%')      /* Until '%' or '\0' */
49                {
50                  if (!format_flag)
51                    {
52                      va_end (ap);
53                      return (0);
54                    }
55                  putchar (format_flag);
56                }
57              switch (format_flag = *format++)
58                {
59                  case 'c':
60                    format_flag = va_arg(ap, int);
61                  default:
62                    putchar (format_flag);
63                    continue;
64
65                  case 's':
66                    ptr = va_arg(ap,char *);
67                    while (format_flag = *ptr++)
68                      {
69                        putchar (format_flag);
70                      }
71                    continue;
72
73                  case 'o':
74                    base = 8;
75                    if (sizeof(int) == 2)
76                      div_val = 0x8000;
77                    else
78                      div_val = 0xC0000000;
79                    goto CONVERSION_LOOP;
80
81                  case 'd':
82                    base = 10;
83                    if (sizeof(int) == 2)
84                      div_val = 10000;
85                    else
86                      div_val = 1000000000;
```

```
87                  goto CONVERSION_LOOP;
88
89                case 'x':
90                  base = 16;
91                  if (sizeof(int) == 2)
92                    div_val = 0x1000;
93                  else
94                    div_val = 0x10000000;
95                  goto CONVERSION_LOOP;
96
97                case 'l':
98                  base = 10;
99                  ldiv_val = 1000000000;
100
101                 lu_val = va_arg(ap,long);
102                 if (format_flag == 'l')
103                   {
104                     if (((long)lu_val) < 0)
105                       {
106                         lu_val = -lu_val;
107                         putchar ('-');
108                       }
109                     while (ldiv_val > 1 && ldiv_val > lu_val)
110                       {
111                         ldiv_val /= 10;
112                       }
113                   }
114                 do
115                   {
116                     putchar (hex[lu_val / ldiv_val]);
117                     lu_val %= ldiv_val;
118                     ldiv_val /= base;
119                   }
120                 while (ldiv_val);
121                 continue;
122
123
124                 CONVERSION_LOOP:
125                   u_val = temp = va_arg(ap,int);
126                   if (format_flag == 'd')
127                     {
128                       if (temp < 0)
129                         {
130                           u_val = -temp;
131                           putchar ('-');
132                         }
133                       while (div_val > 1 && div_val > u_val)
134                         {
135                           div_val /= 10;
136                         }
```

```
                    }
                  do
                    {
                      putchar (hex[u_val / div_val]);
                      u_val %= div_val;
                      div_val /= base;
                    }
                  while (div_val);
                }
              }
            }
```

Errors: none
Warnings: none
Code size: 708

```
Archimedes 8096/80C196 C-Compiler V3.00I/DOS                    13/Jul/92  04:05:02

        Target option =  80C196
        Memory model  =  large
        Source file   =  printf.c
        List file     =  printf.lst
        Object file   =  printf.rl5
        Command line  =  printf.c -ml -vl -z -P -e -L -F -p52
                                                  (c) Copyright Archimedes Software Inc. 1990
```

- PRINTF.C -

```
1          /*
2             The ANSI "printf" function.
3
4             Version: 3.00 [IANR]
5
6          */
7
8          #include "stdarg.h"
9          #include "stdio.h"
10         #include "icclbutl.h"
11
12         static void put_one_char(char c, void *dummy)
13           {
14             putchar (c);
15                        /* Warning on this line OK (unused 'dummy') */
16           }
```

```
                                                    /* Our main entry */
int printf (const char *format, ...)
  {
    va_list ap;
    int nr_of_chars;

    va_start (ap, format);          /* Variable argument begin */
    nr_of_chars = _formatted_write (format, put_one_char, (void *) 0, ap);
    va_end (ap);                    /* Variable argument end */
    return (nr_of_chars);           /* According to ANSI */
  }
```

- printf -

```
    29

Errors: none
Warnings: none
Code size: 63
```

TIMERS.ASM

```
Archimedes 8096/196 Assembler V1.95/MD2          13/Jul/92  12:55:26

Source   =  timers.asm
List     =  timers.lst
Object   =  timers.obj
Options  =  s
                                  (c) Copyright Archimedes Software Inc. 1988

;*****************************************************************************
;*                    S O L A R   F L U X   M O N I T O R                    *
;*                                                                           *
;*              A D V A N C E D   L A N G M U I R   P R O B E                *
;*                                                                           *
;*                          S F M / A L P                                    *
;*                                                                           *
;*                                                                           *
;*              Timer Initialization & Service Routines                      *
;*                                                                           *
;*****************************************************************************

                Name    Timers
                Rseg    Timers
0000
0000
                                Assembler Directives

;               LSTCOD  +
                LSTWID  -
0000            LSTPAG  +
0000            LSTXRF
0000            PAGSIZ  52
0000
0000
0000
0000
```

30
31
32

```
                    ;               ***********************************************
                    ;               *                                             *
                    ;               *                                             *
                    ;               *                                             *
                    ;               *                P U B L I C S                *
                    ;               *                                             *
                    ;               *                                             *
                    ;               *                                             *
                    ;               ***********************************************
                    ;
                    ;
                                     Init_Timer1, Timer1, Timer2
                     Public
0000
```

```
;       ***************************************************************
;       *                                                             *
;       *                                                             *
;       *                   E X T E R N A L S                         *
;       *                                                             *
;       *                                                             *
;       ***************************************************************


```

```
                ;               ***********************************************
                ;               *                                             *
                ;               *      R E G I S T E R   D E F I N I T I O N S *
                ;               *                                             *
                ;               ***********************************************
                ;
                                                        ;INCLUDE FILE FOR 8096 REGISTERS
                $REG8096.INC
                ;       - REG8096.INC -
                ;
                ;       Include file containing 8096 register definitions
                ;
                ;
                ;       8096 C Compiler Working registers
                ;
                IR_BASE DEFINE  01AH            ; Register base address
001A
                R8      DEFINE  (IR_BASE)
001A            R9      DEFINE  (IR_BASE+1)
001B
                R0      DEFINE  (IR_BASE+2)
001C            R1      DEFINE  (IR_BASE+3)
001D            R2      DEFINE  (IR_BASE+4)
001E            R3      DEFINE  (IR_BASE+5)
001F            R4      DEFINE  (IR_BASE+6)
0020            R5      DEFINE  (IR_BASE+7)
0021            R6      DEFINE  (IR_BASE+8)
0022            R7      DEFINE  (IR_BASE+9)
0023
                ;       Special Function Registers
                ;
                ;               DEFINE  18H
0018            SP              DEFINE  18H
0018            STACKPOINTER
                AD_RESULT_LO    DEFINE  02H
0002            AD_RESULT_HI    DEFINE  03H
0003            AD_COMMAND      DEFINE  02H
0002
                HSI_MODE        DEFINE  03H
0003            HSI_TIME        DEFINE  04H
0004            HSI_STATUS      DEFINE  06H
0006
                HSO_TIME        DEFINE  04H
0004            HSO_COMMAND     DEFINE  06H
0006
```

```
                SBUF            DEFINE  07H
0007            SBUF_RX         DEFINE  07H
0007            SBUF_TX         DEFINE  07H
0007

                INT_MASK        DEFINE  08H
0008            INT_PEND        DEFINE  09H
0009

                WATCHDOG        DEFINE  0AH
000A

                TIMER1          DEFINE  0AH
000A            TIMER2          DEFINE  0CH
000C

                BAUD_RATE       DEFINE  0EH
000E

                IO_PORT0        DEFINE  0EH
000E            IO_PORT1        DEFINE  0FH
000F            IO_PORT2        DEFINE  10H
0010

                SP_STAT         DEFINE  11H
0011            SP_CON          DEFINE  11H
0011

                IOS0            DEFINE  15H
0015            IOS1            DEFINE  16H
0016

                IOC0            DEFINE  15H
0015            IOC1            DEFINE  16H
0016

                PWM_CONTROL     DEFINE  17H
0017

                $DEF8096.INC
                ;       - DEF8096.INC -
                ;
                ;       Include file for getting a nonbanked model
                ;
                ;

                banking         DEFINE  0
0000


                $SFMALP.INC                     ; 80C196 Special Function Registers

                INT_PEND1       DEFINE          12h
0012            INT_MASK1       DEFINE          13h
0013            WSR             DEFINE          14h
0014            IOS2            DEFINE          17h
0017            IOC2            DEFINE          17h
0017

                                                ; Registers to be used in assembly routines
                                                1An
                Ax              DEFINE
001A
```

```
001B            Ah              DEFINE          1Bh
001A            Al              DEFINE          1Ah
001C            Bx              DEFINE          1Ch
001D            _Bh             DEFINE          1Dh
001C            Bl              DEFINE          1Ch
001E            Cx              DEFINE          1Eh
001F            Ch              DEFINE          1Fh
001E            Cl              DEFINE          1Eh
0020            Dx              DEFINE          20h
0021            Dh              DEFINE          21h
0020            Dl              DEFINE          20h
0022            Ex              DEFINE          22h
0023            Eh              DEFINE          23h
0022            El              DEFINE          22h
0024            Fx              DEFINE          24h
0025            Fh              DEFINE          25h
0024            Fl              DEFINE          24h

                                                00h ; To be used in comparisons
0000            ZERO            DEFINE

                                                ; Serial Port, Jiffiy Clock, and Flags
                                                26h
0026            Rcv_In_Ptr      DEFINE          28h
0028            Rcv_Out_Ptr     DEFINE          2Ah
002A            Xmt_In_Ptr      DEFINE          2Ch
002C            Xmt_Out_Ptr     DEFINE

                                                2Eh
002E            Jiffy_Clock     DEFINE

                                                30h
0030            Serial_Flags    DEFINE          31h
0031            Flags           DEFINE          32h
0032            Flags2          DEFINE
```

```
                ;               ***************************************************
                ;               *                                                 *
                ;               *                                                 *
                ;               *                  E Q U A T E S                  *
                ;               *                                                 *
                ;               *                                                 *
                ;               ***************************************************
                ;

                                                ;   Serial Flag Register bit locations
                                                        ; Bit 0 = Xmt Active
0000            Xmt_Active:     equ     0               ; Bit 1 = Xmt Full
0001            Xmt_Full:       equ     1               ; Bit 2 = Xmt Empty
0002            Xmt_Empty:      equ     2               ; Bit 3 = Rcv Full
0003            Rcv_Full:       equ     3               ; Bit 4 = Rcv Empty
0004            Rcv_Empty:      equ     4               ; Bit 5 = CR Detected
0005            CR_Detected:    equ     5               ; Bit 6 =
                                                        ; Bit 7 =

                                                ;   Flags register assignment
                                                        ; Bit 0 = Timer 1 Interrupt Occured
0000            Timer1_Int:     equ     0               ; Bit 1 = Software Enable of Jiffy Timer
0001            Jiffy_Override: equ     1               ; Bit 2 =
                                                        ; Bit 3 =
                                                        ; Bit 4 =
                                                        ; Bit 5 =
                                                        ; Bit 6 =
                                                        ; Bit 7 =

                                                ; Flags 2 Register assignment
                                                        ; Bit 0 =
                                                        ; Bit 1 =
                                                        ; Bit 2 =
                                                        ; Bit 3 =
                                                        ; Bit 4 =
                                                        ; Bit 5 =
                                                        ; Bit 6 =
                                                        ; Bit 7 =

                                                ;   IO_PORT2 pin assignments (input port)
                                                        ; Bit 0 =
                                                        ; Bit 1 =
                                                        ; Bit 2 =
                                                        ; Bit 3 =
0003            Accel:          equ     3
```

```
                T_Full:         equ     4       ; Bit 4 =
0004                                            ; Bit 5 =
                                                ; Bit 6 =
                                                ; Bit 7 =
                                                ; ASCII characters
                Cr:             equ     13      ; Carriage Return
000D            Lf:             equ     10      ; Line Feed
000A            Bs:             equ     08      ; Back Space
0008            Del:            equ     7Fh     ; Delete
007F            Bel:            equ     07      ; Bell
0007            FF:             equ     12      ; Form Feed
000C            Ht:             equ     9       ; Horizontal Tab
0009            Vt:             equ     11      ; Vertical Tab
000B            Sp:             equ     32      ; Space
0020
                Port1_init:             equ     000h    ; No Instrument Selected
0000            Port2_init:             equ     0FFh    ; All Inputs HIGH
00FF
```

258

```
;               ***********************************************
;               *                                             *
;               *                                             *
;               *        H S O   U N I T   T I M E R S        *
;               *                                             *
;               *                                             *
;               ***********************************************
;
;       Timer1  ==  5 mSec jiffy clock
;       Timer2  ==  Flash ring timer
;
;       Init_Timer1             :       Initialize timer 1
;       Timer1                  :       Timer1 handler routine
;
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                    I N I T   T I M E R   1                    *
                ;*                                                               *
                ;*****************************************************************
                ;* inputs    |   outputs   |   routines called   |  registers altered *
                ;*****************************************************************
                ;*   none    |    none     |       Timer1        |        Ax          *
                ;*****************************************************************

                ;       This routine initializes the High Speed Output unit into
                ;       generating Timer 1 interrupts.  It uses the Timer 1 interrupt routine
                ;       to actually command the timer to start.


                Init_Timer1:

0000                    ld      Ax, TIMER1                      ; read present time into Ax
0000 A00A1A             orb     INT_MASK, #00100000b            ; enable Software timer int.
0003 912008             call    Timer1                          ; call Timer1 ISR.
0006 EF0200             ei                                      ; enable interrupts
0009 FB                 ret
000A F0
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                  T I M E R 1                              *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |   outputs    |   routines called   |   registers altered  *
                ;*****************************************************************************
                ;*    Ax       |  HSO Command |        none         |        none          *
                ;*             |     flags    |                     |                      *
                ;*****************************************************************************

                ;               This routine set the HSO unit to generate a timer1 interrupt
                ;       X mSec from the time passed in Ax.  It also sets a flag to indicate
                ;       that a timer1 interrupt occured.
                ;               Once every interrupt the Flash status bit is complimented.
                ;               The Accelerometer launch input is tested and if high the
                ;       jiffy clock is incremented. If low then the clock is set to zero.
                ;       If the jiffy_override bit is set the accel input is not tested
                ;       and the clock is incremented.

                ; Constants: for 8096   mS * 500 + 3
                ;            for 80196  mS * 750 + 3
                ;

                Timer1_Const_96:        equ     2500 + 3        ; state counts required for
                Timer1_Const_196:       equ     3750 + 3        ;       4 mSec delay


                Timer1:         Equ     $

000B                    ldb     HSO_COMMAND, #00111000b         ; send timer1 HSO Command
                                                                ;   enable HSO interrupt
000B B13806                                                     ;   Set flag = enabled


                                                                ; Set Time out constand. Assume
                                                                ;   3 HSO counts elasped before
                                                                ;   Ax = Timer1 time. ignore
                                                                ;   carry and overflow.

                        Add     Ax, #Timer1_Const_196
000E 65A90E1A           ld      HSO_TIME, Ax                    ; Set Timer1_Int flag
0012 A01A04             orb     Flags, #00000001b
0015 910131                                                     ; increment clock time.
                        inc     Jiffy_Clock                     ; If override set then run clock
0018 072E               jbs     Flags, Jiffy_Override, T_20     ; If Accel = 1 then run clock
001A 393105             jbs     IO_PORT2, Accel, T_20           ;   else clock time = 0.
001D 3B1002             clr     Jiffy_Clock
0020 012E

0022 F0         T_20:   ret
```

360

```
                ;*****************************************************************
                ;*                                                               *
                ;*                          T I M E R 2                          *
                ;*                                                               *
                ;*****************************************************************
                ;* inputs     |   outputs    |   routines called   |  registers altered *
                ;*****************************************************************
                ;*    none    |    none      |    none             |       none         *
                ;*****************************************************************

                Timer2:
                        ret

 0023                   END
 0023 F0
 0024
```

Errors: None
Bytes: 36
CRC: 815F

Timers

SFM/ALP TimerService Routines

Symbol and Cross Reference Table

Symbol	Value	Type	Defline	Refline			
Segment Definitions							
Timers	0024	S00R	15				
External Symbols							
Public Symbols							
Init_Timer1	0000	R00E	299	43			
Timer1	000B	R00E	341	43	303		
Timer2	0023	R00E	375	43			
Local Symbols							
AD_COMMAND	0002	A‡	97				
AD_RESULT_HI	0003	A‡	96				
AD_RESULT_LO	0002	A‡	95				
Accel	0003	A	236	356			
Ah	001B	A‡	155				
Al	001A	A‡	156				
Ax	001A	A‡	154	301	350	351	
BAUD_RATE	000E	A‡	118				

Bel	0007	A	248		
Bl	001C	A‡	159		
Bs	0008	A	246		
Bx	001C	A‡	157		
CR_Detected	0005	A	205		
Ch	001F	A‡	161		
Cl	001E	A‡	162		
Cr	000D	A	244		
Cx	001E	A‡	160		
Del	007F	A	247		
Dh	0021	A‡	164		
Dl	0020	A‡	165		
Dx	0020	A‡	163		
Eh	0023	A‡	167		
El	0022	A‡	168		
Ex	0022	A‡	166		
FF	000C	A	249		
Fh	0025	A‡	170		
Fl	0024	A‡	171		
Flags	0031	A‡	184	352	355
Flags2	0032	A‡	185		
Fx	0024	A‡	169		
HSI_MODE	0003	A‡	99		
HSI_STATUS	0006	A‡	101		

SERIAL.ASM

```
#################################################################
#                                                               #
#     Archimedes 8096/196 Assembler V1.95/MD2        13/Jul/92  04:04:03   #
#                                                               #
#      Source   =   serial.asm                                  #
#      List     =   serial.lst                                  #
#      Object   =   serial.obj                                  #
#      Options  =   s                                           #
#                                                               #
#                              (c) Copyright Archimedes Software Inc. 1988 #
#################################################################
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                                                               *
                ;*                                                               *
                ;*                        S E R I A L . a s m                    *
                ;*                                                               *
                ;*                                                               *
                ;*                                                               *
                ;*                                                               *
                ;*            Interrupt Serial Port Driver routines for DATES.asm *
                ;*            program.                                           *
                ;*                                                               *
                ;*****************************************************************


0000                    Name    Ser
0000                    Rseq    Ser


                ;                       Assembler Directives
0000                    LSTCOD  +
0000                    LSTWID  -
0000                    LSTPAG  +
0000                    LSTXRF
0000                    PAGSIZ  56
```

30

```
                $REG8096.INC
                ;       - REG8096.INC -
                ;
                ;       Include file containing 8096 register definitions
                ;

                ;
                ;       8096 C Compiler Working registers
                ;
001A            IR_BASE DEFINE  01AH            ; Register base address

001A            R8      DEFINE  (IR_BASE)
001B            R9      DEFINE  (IR_BASE+1)

001C            R0      DEFINE  (IR_BASE+2)
001D            R1      DEFINE  (IR_BASE+3)
001E            R2      DEFINE  (IR_BASE+4)
001F            R3      DEFINE  (IR_BASE+5)
0020            R4      DEFINE  (IR_BASE+6)
0021            R5      DEFINE  (IR_BASE+7)
0022            R6      DEFINE  (IR_BASE+8)
0023            R7      DEFINE  (IR_BASE+9)

                ;
                ;       Special Function Registers
                ;
0018            SP              DEFINE  18H
0018            STACKPOINTER    DEFINE  18H

0002            AD_RESULT_LO    DEFINE  02H
0003            AD_RESULT_HI    DEFINE  03H
0002            AD_COMMAND      DEFINE  02H

0003            HSI_MODE        DEFINE  03H
0004            HSI_TIME        DEFINE  04H
0006            HSI_STATUS      DEFINE  06H

0004            HSO_TIME        DEFINE  04H
0006            HSO_COMMAND     DEFINE  06H

0007            SBUF            DEFINE  07H
0007            SBUF_RX         DEFINE  07H
0007            SBUF_TX         DEFINE  07H

0008            INT_MASK        DEFINE  08H
0009            INT_PEND        DEFINE  09H

000A            WATCHDOG        DEFINE  0AH

000A            TIMER1          DEFINE  0AH
000C            TIMER2          DEFINE  0CH

000E            BAUD_RATE       DEFINE  0EH
```

```
 85
 86  000E        IO_PORT0        DEFINE  0EH
 87  000F        IO_PORT1        DEFINE  0FH
 88  0010        IO_PORT2        DEFINE  10H
 89
 90  0011        SP_STAT         DEFINE  11H
 91  0011        SP_CON          DEFINE  11H
 92
 93  0015        IOS0            DEFINE  15H
 94  0016        IOS1            DEFINE  16H
 95
 96  0015        IOC0            DEFINE  15H
 97  0016        IOC1            DEFINE  16H
 98
 99  0017        PWM_CONTROL     DEFINE  17H
100
101              $DEF8096.INC
102              ;       - DEF8096.INC -
103              ;
104              ;       Include file for getting a nonbanked model
105              ;
106
107  0000        banking         DEFINE  0
108
109
110
111              $SFMALP.INC
112                                          ; 80C196 Special Function Registers
113  0012        INT_PEND1       DEFINE      12h
114  0013        INT_MASK1       DEFINE      13h
115  0014        WSR             DEFINE      14h
116  0017        IOS2            DEFINE      17h
117  0017        IOC2            DEFINE      17h
118
119                                          ; Registers to be used in assembly routines
120  001A        Ax              DEFINE      1Ah
121  001B        Ah              DEFINE      1Bh
122  001A        Al              DEFINE      1Ah
123  001C        Bx              DEFINE      1Ch
124  001D        _Bh             DEFINE      1Dh
125  001C        Bl              DEFINE      1Ch
126  001E        Cx              DEFINE      1Eh
127  001F        Ch              DEFINE      1Fh
128  001E        Cl              DEFINE      1Eh
129  0020        Dx              DEFINE      20h
130  0021        Dh              DEFINE      21h
131  0020        Dl              DEFINE      20h
132  0022        Ex              DEFINE      22h
133  0023        Eh              DEFINE      23h
134  0022        El              DEFINE      22h
135  0024        Fx              DEFINE      24h
136  0025        Fh              DEFINE      25h
137  0024        Fl              DEFINE      24h
```

```
0000        ZERO            DEFINE      00h ; To be used in comparisons

                                        ; Serial Port, Jiffly Clock, and Flags
0026        Rcv_In_Ptr      DEFINE      26h
0028        Rcv_Out_Ptr     DEFINE      28h
002A        Xmt_In_Ptr      DEFINE      2Ah
002C        Xmt_Out_Ptr     DEFINE      2Ch

002E        Jiffy_Clock     DEFINE      2Eh

0030        Serial_Flags    DEFINE      30h
0031        Flags           DEFINE      31h
0032        Flags2          DEFINE      32h
```

```
154
155                 ;               ***************************************************
156                 ;               *                                                 *
157                 ;               *                                                 *
158                 ;               *                    E Q U A T E S                *
159                 ;               *                                                 *
160                 ;               *                                                 *
161                 ;               ***************************************************
162
163
164
165                                         ;   Serial Flag Register bit locations
166  0000           Xmt_Active:     equ     0       ; Bit 0 = Xmt Active
167  0001           Xmt_Full:       equ     1       ; Bit 1 = Xmt Full
168  0002           Xmt_Empty:      equ     2       ; Bit 2 = Xmt Empty
169  0003           Rcv_Full:       equ     3       ; Bit 3 = Rcv Full
170  0004           Rcv_Empty:      equ     4       ; Bit 4 = Rcv Empty
171  0005           CR_Detected:    equ     5       ; Bit 5 = CR Detected
172                                                 ; Bit 6 =
173                                                 ; Bit 7 =
174
175
176                                         ;   Flags register assignment
177  0000           Timer1_Int:     equ     0       ; Bit 0 = Timer 1 Interrupt Occured
178  0001           Jiffy_Override: equ     1       ; Bit 1 = Software Enable of Jiffy Timer
179                                                 ; Bit 2 =
180                                                 ; Bit 3 =
181                                                 ; Bit 4 =
182                                                 ; Bit 5 =
183                                                 ; Bit 6 =
184                                                 ; Bit 7 =
185
186
187                                         ; Flags 2 Register assignment
188                                                 ; Bit 0 =
189                                                 ; Bit 1 =
190                                                 ; Bit 2 =
191                                                 ; Bit 3 =
192                                                 ; Bit 4 =
193                                                 ; Bit 5 =
194                                                 ; Bit 6 =
195                                                 ; Bit 7 =
196
197
198                                         ;  IO_PORT2 pin assignments (input port)
199                                                 ; Bit 0 =
200                                                 ; Bit 1 =
201                                                 ; Bit 2 =
202  0003           Accel:          equ     3       ; Bit 3 =
203  0004           T_Full:         equ     4       ; Bit 4 =
204                                                 ; Bit 5 =
205                                                 ; Bit 6 =
206                                                 ; Bit 7 =
```

```
                                        ; ASCII characters
000D        Cr:             equ     13      ; Carriage Return
000A        Lf:             equ     10      ; Line Feed
0008        Bs:             equ     08      ; Back Space
007F        Del:            equ     7Fh     ; Delete
0007        Bel:            equ     07      ; Bell
000C        FF:             equ     12      ; Form Feed
0009        Ht:             equ     9       ; Horizontal Tab
000B        Vt:             equ     11      ; Vertical Tab
0020        Sp:             equ     32      ; Space

0000        Port1_init:             equ     000h    ; No Instrument Selected
00FF        Port2_init:             equ     0FFh    ; All Inputs HIGH
```

224

```
                ;               ************************************************
                ;               *                                              *
                ;               *                                              *
                ;               *                E X T E R N A L S   &         *
                ;               *                    P U B L I C S             *
                ;               *                                              *
                ;               ************************************************



                                ; Equate Publics

0000            Extrn           Rcv_Buffer, Xmt_Buffer, Rcv_End, Xmt_End

0000            Public          Init_Serial, Rx_Int, Tx_int
0000            Public          Rcv_Char, Xmt_Char, Send_Msg, Wait_Tx_Inactive
0000            Public          Send_Hex_Word, Send_Hex_Byte, Asc_Hex, Wait_CR
0000            Public          Clear_Com
```

```
;               ***************************************************
;               *                                                 *
;               *     B U F F E R E D   /   I N T E R R U P T     *
;               *                                                 *
;               *     D R I V E N   S E R I A L   P O R T         *
;               *                                                 *
;               *                R O U T I N E S                  *
;               *                                                 *
;               ***************************************************

;       Init_Serial     :       Initializes serial port and buffers
;       Rcv_Char        :       Reads 1 character from serial port
;       Xmt_Char        :       Sends 1 character out serial port
;       Send_Msg        :       Sends a message out serial port
;       Send_Hex_Word   :       Sends word in Ax as ascii encoded hex
;       Send_Hex_Byte   :       Sends byte in Al as ascii encoded hex
;       Wait_CR         :       Waits until a carriage return is detected
;       Wait_Tx_Inactive:       Waits until the serial Xmt is inactive
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                          I N I T    S E R I A L                           *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs    |    routines called    |   registers altered *
                ;*****************************************************************************
                ;*   none      |     Baud      |        none           |      INT_MASK       *
                ;*             |    SP_CON     |                       |                     *
                ;*             |    In_Ptr     |                       |                     *
                ;*             |    Out_Ptr    |                       |                     *
                ;*             |  Serial_Flags |                       |                     *
                ;*****************************************************************************

                ;         This routine initializes the serial interrupts and input/output
                ;      buffer pointers.

0000            Init_Serial:


                                                        ;   xtal = clk source
0000 B14D0E          ldb     BAUD_RATE, # Low  804Dh ; Baud rate for 80c196
0003 B1800E          ldb     BAUD_RATE, # High 804Dh ;   xtal = clock source.
                                                        ; Set receive buffer pointers
0006 B10911          ldb     SP_CON, #00001001b      ; No Parity
                                                        ; Receiver enabled
                                                        ; serial port mode 1.
0009 A1000026        ld      Rcv_In_Ptr, # Rcv_Buffer
000D A1000028        ld      Rcv_Out_Ptr, # Rcv_Buffer
                                                        ; Set transmitt buffer pointers
0011 A100002A        ld      Xmt_In_Ptr, # Xmt_Buffer
0015 A100002C        ld      Xmt_Out_Ptr, # Xmt_Buffer


0019 B11430          ldb     Serial_Flags, #00010100b        ; Clear flags register
                                                             ; Rcv = Empty
                                                             ; Xmt = Empty

001C 914008          orb     INT_MASK, #01000000b    ; enable serial interrupt
001F FB              ei                              ; enable interrupts
0020 F0              ret
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                                R X     I N T                              *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;* inputs      |    outputs    |    routines called    |   registers altered *
                ;*****************************************************************************
                ;*   SBUF      |   Rcv_Buff    |                       |     Rcv_In_Ptr      *
                ;*             |               |                       |                     *
                ;*             | Serial_Flags  |                       |                     *
                ;*             |               |                       |                     *
                ;*****************************************************************************

                ;               This routine completes the handling of a Serial Recieve
                ;       Interrupt.  It is entered by a jump from the Serial_ISR.


0021            Rx_Int:                 ; Handle Receive interrupt

0021 B0071B             ldb     Ah, SBUF                ; Read Character from serial port

0024 99031B             cmpb    Ah, #03                 ; ck for <cntl> C
0027 D701               jne     RI_00                   ;   reset processor on control C
0029 FF                 rst

002A 99131B     RI_00:  cmpb    Ah, #19                 ; ck for <cntl> S
002D D707               jne     RI_01
002F 71EF31             andb    Flags, #0EFh            ; disable serial telem
0032 CC1A               pop     Ax                      ; Exit ISR
0034 F3                 popf
0035 F0                 ret

0036 997F1B     RI_01:  cmpb    Ah, #Del                ; Check for delete or backspace
0039 DF05               je      RI_10
003B 99081B             cmpb    Ah, #Bs
003E D73F               jne     RI_20                   ; jump if neither backspace or delete.

0040 B1081A     RI_10:  ldb     Al, #Bs                 ; If received delete or backspace send
0043 EFD500             call    Xmt_Char                ;   Back space, Space, Back space
0046 B1201A             ldb     Al, #' '
0049 EFCF00             call    Xmt_Char
004C B1081A             ldb     Al, #Bs
004F EFC900             call    Xmt_Char

0052 3C3008             jbs     Serial_Flags, Rcv_Empty, RI_11 ; if buff empty. don't remove
                                                               ;   any more characters.
0055 3B300C             jbs     Serial_Flags, Rcv_Full, RI_12  ; if buffer not full and
0058 882826             cmp     Rcv_In_Ptr, Rcv_Out_Ptr        ;   (in ptr = out_ptr) then
005B D707               jne     RI_12                          ;   receive buffer empty.
```

```
                                          modes2
                                          tm
                                          modes_tm
                                          io
          SFM_Flight_Mode        B5C8     Not referred to
          ALP_Flight_Mode        B7F4     Not referred to

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

  FILE NAME : modes2.rl5
  PROGRAM MODULE, NAME : modes2

  SEGMENTS IN THE MODULE
  ======================
CODE
  Relative segment, address : 5DA4 - 6881
          ENTRIES                ADDRESS        REF BY MODULE
          sfm_std                 5DA4          modes
          sfm_irpa                6170          modes
          sfm_hi_res              5E43          modes
          sfm_gt_3                6096          modes
          sfm_gt_2                5FBC          modes
          sfm_gt_1                5EE2          modes
          sfm_erpa                620F          modes
          sfm_clean               62B6          modes
          alp_std                 6636          modes
          alp_m_res               6725          modes
          alp_hr_ne               67CB          modes
          alp_hi_res              66AC          modes
          alp_clean               6805          modes
          alp_5_pt                6340          modes
    ---------------------------------------------------------------------------
UDATA
  Relative segment, address : D1CE - D1E5
          ENTRIES                ADDRESS        REF BY MODULE
          v5_conv                 D1E0          sfm-alp
                                                com_proc
                                                modes
                                                tm
                                                modes_tm
                                                io
          v4_conv                 D1DC          sfm-alp
                                                com_proc
                                                modes
                                                tm
                                                modes_tm
                                                io
          v3_conv                 D1D8          sfm-alp
                                                com_proc
                                                modes
                                                tm
                                                modes_tm
                                                io
          v2_conv                 D1D4          sfm-alp
                                                com_proc
                                                modes
```

		tm
		modes_tm
		io
kTe	D1E4	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io
i5_conv	D1E2	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io
i4_conv	D1DE	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io
i3_conv	D1DA	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io
i2_conv	D1D6	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io
i1_real	D1D0	Not referred to
i1_conv	D1CE	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io

CDATA

Relative segment, address : A71E - A71F

IDATA

Relative segment, address : D152 - D153

ENTRIES	ADDRESS	REF BY MODULE
v1_conv	D152	sfm-alp
		com_proc
		modes
		tm
		modes_tm
		io

****

FILE NAME : tm.rl5
PROGRAM MODULE, NAME : tm

SEGMENTS IN THE MODULE
==========

CODE
Relative segment, address : 6882 - 6FF6

ENTRIES	ADDRESS	REF BY MODULE
test_telem_two_chan	6C49	Not referred to
test_telem	6958	com_proc
send_telem	6F84	Not referred to
send_ready_tm	68ED	sfm-alp
init_telem	6DDB	sfm-alp
data_to_tm	6882	sfm-alp
		com_proc

CSTR
Relative segment, address : B2F2 - B2FE

UDATA
Relative segment, address : D1E6 - D1F1

ENTRIES	ADDRESS	REF BY MODULE
send_serial_tm	D1EA	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
new_SFM_AD_val	D1EE	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
new_ALP_AD_val	D1EC	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
idle_count	D1F0	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
SFM_packet_ready	D1E7	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
IDLE_packet_ready	D1E9	sfm-alp
		com_proc
		modes
		modes2
		modes_tm
HSKP_packet_ready	D1E8	sfm-alp
		com_proc
		modes
		modes2

```
                                                modes_tm
          ALP_packet_ready        D1E6          sfm-alp
                                                com_proc
                                                modes
                                                modes2
                                                modes_tm

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

  FILE NAME : modes_tm.rl5
  PROGRAM MODULE, NAME : modes_tm

  SEGMENTS IN THE MODULE
  ======================
CODE
  Relative segment, address : 6FF8 - 7AD9
          ENTRIES                ADDRESS          REF BY MODULE
          tm_sfm_std              6FF8            Not referred to
          tm_sfm_irpa             7433            Not referred to
          tm_sfm_hi_res           70AA            Not referred to
          tm_sfm_gt_3             7341            Not referred to
          tm_sfm_gt_2             724F            Not referred to
          tm_sfm_gt_1             715D            Not referred to
          tm_sfm_erpa             74E5            Not referred to
          tm_sfm_clean            7597            Not referred to
          tm_null_mode            7AD8            Not referred to
          tm_hskp_mode            7AD9            Not referred to
          tm_alp_std              7774            Not referred to
          tm_alp_m_res            78D9            Not referred to
          tm_alp_hr_ne            799B            Not referred to
          tm_alp_hi_res           7826            Not referred to
          tm_alp_clean            7A4E            Not referred to
          tm_alp_5_pt             7621            Not referred to
    -----------------------------------------------------------------------
CONST
  Relative segment, address : BB48 - BB67
          ENTRIES                ADDRESS          REF BY MODULE
          tm_fun_list             BB48            tm

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

  FILE NAME : crc.rl5
  PROGRAM MODULE, NAME : crc

  SEGMENTS IN THE MODULE
  ======================
CODE
  Relative segment, address : 7ADA - 7D14
          ENTRIES                ADDRESS          REF BY MODULE
          crc_word                7B53            Not referred to
          crc_byte                7ADA            Not referred to
          crc_buff                7C5A            Not referred to

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

  FILE NAME : io.rl5
```

PROGRAM MODULE, NAME : io

SEGMENTS IN THE MODULE
==========

CODE
Relative segment, address : 7D16 - 82D0

ENTRIES	ADDRESS	REF BY MODULE
uncln_prb	81C5	modes modes2
shift_oce_vals	7D47	com_proc tm calib
set_vern_pg	7FEF	com_proc modes calib
set_va_dac	7FB4	modes modes2 tm
set_va	7EFA	sfm-alp com_proc modes2 tm calib
set_grid_dac	7E84	tm
set_grid	7DAA	sfm-alp com_proc modes2 tm
set_dac4	7ED8	modes tm
read_ad	82C6	Not referred to
init_oces	7D16	sfm-alp
clean_prb	80C5	modes2

CSTR
Relative segment, address : B300 - B344

****

FILE NAME : calib.rl5
PROGRAM MODULE, NAME : calib

SEGMENTS IN THE MODULE
==========

CODE
Relative segment, address : 82D2 - 870A

ENTRIES	ADDRESS	REF BY MODULE
c_va_sweep	8623	com_proc
c_calib_vr	8551	com_proc
c_calib_va	82D2	com_proc
c_calib_gr	836C	com_proc
c_calib_em	8427	com_proc

CSTR
Relative segment, address : B346 - B41A

****

FILE NAME : sfmalp.obj
PROGRAM MODULE, NAME : sfmalp

ABSOLUTE ENTRIES	ADDRESS	REF BY MODULE
Probe_Cln_WR	F000	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Telem_WR	FC00	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Telem	FC00	Test
EMeter_RD	F400	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
EMeter_WR	F800	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
VAGen_WR	F800	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Timer_WR	F000	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Grid_Volt_WR	F000	sfm-alp

		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
loader	2100	com_proc
Loader_CS	4000	Test
EEPROM_stop	C000	Test
EEPROM_start	4000	Test
ZRAM_stop	DFFF	Test
ZRAM_start	C000	Test
Flags2	0032	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Flags	0031	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Serial_Flags	0030	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Fl	0024	Not referred to
Fh	0025	Not referred to
Fx	0024	Not referred to
El	0022	Not referred to
Eh	0023	Not referred to
Ex	0022	Not referred to
Dl	0020	Not referred to
Dh	0021	Not referred to
Dx	0020	Not referred to
Cl	001E	Not referred to
Ch	001F	Not referred to
Cx	001E	Not referred to
Bl	001C	Not referred to
_Bh	001D	Not referred to
Bx	001C	Not referred to
Al	001A	Not referred to
Ah	001B	Not referred to
Ax	001A	Not referred to

ABSOLUTE LOCALS	ADDRESS
IR_BASE	001A
R8	001A
R9	001B
R0	001C
R1	001D
R2	001E
R3	001F
R4	0020
R5	0021
R6	0022
R7	0023
SP	0018
STACKPOINTER	0018
AD_RESULT_LO	0002
AD_RESULT_HI	0003
AD_COMMAND	0002
HSI_MODE	0003
HSI_TIME	0004
HSI_STATUS	0006
HSO_TIME	0004
HSO_COMMAND	0006
SBUF	0007
SBUF_RX	0007
SBUF_TX	0007
INT_MASK	0008
INT_PEND	0009
WATCHDOG	000A
TIMER1	000A
TIMER2	000C
BAUD_RATE	000E
IO_PORT0	000E
IO_PORT1	000F
IO_PORT2	0010
SP_STAT	0011
SP_CON	0011
IOS0	0015
IOS1	0016
IOC0	0015
IOC1	0016
PWM_CONTROL	0017
banking	0000
INT_PEND1	0012
INT_MASK1	0013
WSR	0014
IOS2	0017
IOC2	0017
ZERO	0000
Rcv_In_Ptr	0026
Rcv_Out_Ptr	0028
Xmt_In_Ptr	002A
Xmt_Out_Ptr	002C
Jiffy_Clock	002E
Xmt_Active	0000
Xmt_Full	0001

Xmt_Empty	0002
Rcv_Full	0003
Rcv_Empty	0004
CR_Detected	0005
Timer1_Int	0000
Jiffy_Override	0001
Accel	0003
T_Full	0004
Cr	000D
Lf	000A
Bs	0008
Del	007F
Bel	0007
FF	000C
Ht	0009
Vt	000B
Sp	0020
Port1_init	0000
Port2_init	00FF
Delay_Const_96	014C
Delay_Const_196	01F2

SEGMENTS IN THE MODULE
==========

CSTACK
Relative segment, address : C000 - C7FF

LOCALS	ADDRESS
Stack_Ptr	C800

ZRAM
Relative segment, address : D000 - D151

ENTRIES	ADDRESS	REF BY MODULE
clean_word	D01C	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
SFM_mode_start_time	D022	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
ALP_mode_start_time	D01E	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
Curr_SFM_Mode	D008	sfm-alp

		com_proc
		modes
		modes2
		tm
		modes_tm
		io
SFM_Mode_Step	D00C	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
SFM_Mode_Ptr	D004	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
Curr_ALP_Mode	D006	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
ALP_Mode_Step	D00A	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
ALP_Mode_Ptr	D002	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
Xmt_End	D152	Ser
Rcv_End	D076	Ser
Xmt_Buffer	D076	Ser
Rcv_Buffer	D026	Ser
Rom_CS	D000	Test
Clock_2P	D00E	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
ALP_VA_Val	D01A	sfm-alp
		com_proc

		modes
		modes2
		tm
		modes_tm
		io
		calib
SFM_VA_Val	D018	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
ALP_Vern_PG	D016	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
SFM_Vern_PG	D014	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib
Status	D012	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
		io
		calib

Telem_Buff
Relative segment, address : D1F2 - D851

ENTRIES	ADDRESS	REF BY MODULE
IDLE_telem	D832	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
HSKP_telem	D7F2	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm
SFM_telem	D4F2	sfm-alp
		com_proc

		modes
		modes2
		tm
		modes_tm
ALP_telem	D1F2	sfm-alp
		com_proc
		modes
		modes2
		tm
		modes_tm

ROM
Relative segment, address : 4300 - 4381

ENTRIES	ADDRESS	REF BY MODULE
Delay	433A	Test
read_AD	4352	com_proc
sleep	434B	sfm-alp
delay	4336	sfm-alp
		com_proc
		modes
		tm
		io
		calib

LOCALS	ADDRESS
Reset	4300
Rst_10	4325
Main_Program	4331
D_30	433C
D_40	4340
RA_10	436A

****

FILE NAME : serial.obj
PROGRAM MODULE, NAME : Ser

ABSOLUTE LOCALS	ADDRESS
IR_BASE	001A
R8	001A
R9	001B
R0	001C
R1	001D
R2	001E
R3	001F
R4	0020
R5	0021
R6	0022
R7	0023
SP	0018
STACKPOINTER	0018
AD_RESULT_LO	0002
AD_RESULT_HI	0003
AD_COMMAND	0002
HSI_MODE	0003
HSI_TIME	0004
HSI_STATUS	0006

HSO_TIME	0004
HSO_COMMAND	0006
SBUF	0007
SBUF_RX	0007
SBUF_TX	0007
INT_MASK	0008
INT_PEND	0009
WATCHDOG	000A
TIMER1	000A
TIMER2	000C
BAUD_RATE	000E
IO_PORT0	000E
IO_PORT1	000F
IO_PORT2	0010
SP_STAT	0011
SP_CON	0011
IOS0	0015
IOS1	0016
IOC0	0015
IOC1	0016
PWM_CONTROL	0017
banking	0000
INT_PEND1	0012
INT_MASK1	0013
WSR	0014
IOS2	0017
IOC2	0017
Ax	001A
Ah	001B
Al	001A
Bx	001C
_Bh	001D
Bl	001C
Cx	001E
Ch	001F
Cl	001E
Dx	0020
Dh	0021
Dl	0020
Ex	0022
Eh	0023
El	0022
Fx	0024
Fh	0025
Fl	0024
ZERO	0000
Rcv_In_Ptr	0026
Rcv_Out_Ptr	0028
Xmt_In_Ptr	002A
Xmt_Out_Ptr	002C
Jiffy_Clock	002E
Serial_Flags	0030
Flags	0031
Flags2	0032
Xmt_Active	0000
Xmt_Full	0001

Xmt_Empty	0002
Rcv_Full	0003
Rcv_Empty	0004
CR_Detected	0005
Timer1_Int	0000
Jiffy_Override	0001
Accel	0003
T_Full	0004
Cr	000D
Lf	000A
Bs	0008
Del	007F
Bel	0007
FF	000C
Ht	0009
Vt	000B
Sp	0020
Port1_init	0000
Port2_init	00FF

SEGMENTS IN THE MODULE
==========

Ser

Relative segment, address : 4382 - 4654

ENTRIES	ADDRESS	REF BY MODULE
Clear_Com	457C	sfm-alp
		calib
Wait_CR	4574	sfmalp
		ISR
Asc_Hex	455A	sfmalp
		ISR
Send_Hex_Byte	4537	sfmalp
		ISR
Send_Hex_Word	4529	sfmalp
		ISR
		Test
Wait_Tx_Inactive	4578	sfmalp
		ISR
Send_Msg	44F2	sfmalp
		ISR
		Test
Xmt_Char	449D	sfmalp
		ISR
		Test
		putchar
Rcv_Char	4470	sfmalp
		ISR
		getchar
Tx_int	443D	sfmalp
		ISR
Rx_Int	43A3	sfmalp
		ISR
Init_Serial	4382	sfmalp
		ISR
LOCALS	ADDRESS	

RI_00	43AC
RI_01	43B8
RI_10	43C2
RI_11	43DF
RI_12	43E6
RI_13	43F2
RI_14	43FD
RI_20	4401
RI_30	440A
RI_40	441A
RI_50	442E
RI_60	4439
TI_10	4444
Send	4459
TI_20	4459
TI_30	4466
RC_10	447E
RC_20	4489
RC_30	4493
RC_40	449B
XC_10	44B1
XC_20	44B9
XC_30	44C0
Set_Xmt_Active	44C2
Clr_Xmt_Active	44C6
Set_Xmt_Full	44CA
Clr_Xmt_Full	44CE
Set_Xmt_Empty	44D2
Clr_Xmt_Empty	44D6
Set_Rcv_Full	44DA
Clr_Rcv_Full	44DE
Set_Rcv_Empty	44E2
Clr_Rcv_Empty	44E6
Set_CR_Detected	44EA
Clr_CR_Detected	44EE
SM_10	44F8
SM_20	4504
SM_30	4509
SM_40	4518
SM_50	4524
SBH_10	454C
SBH_20	4557
AH_20	456E
AH_10	4570
Message_Table	4586
Msg_Table_End	4655

****

FILE NAME : isr.obj
PROGRAM MODULE, NAME : ISR

ABSOLUTE LOCALS	ADDRESS
==========	=======
IR_BASE	001A
R8	001A
R9	001B

R0	001C
R1	001D
R2	001E
R3	001F
R4	0020
R5	0021
R6	0022
R7	0023
SP	0018
STACKPOINTER	0018
AD_RESULT_LO	0002
AD_RESULT_HI	0003
AD_COMMAND	0002
HSI_MODE	0003
HSI_TIME	0004
HSI_STATUS	0006
HSO_TIME	0004
HSO_COMMAND	0006
SBUF	0007
SBUF_RX	0007
SBUF_TX	0007
INT_MASK	0008
INT_PEND	0009
WATCHDOG	000A
TIMER1	000A
TIMER2	000C
BAUD_RATE	000E
IO_PORT0	000E
IO_PORT1	000F
IO_PORT2	0010
SP_STAT	0011
SP_CON	0011
IOS0	0015
IOS1	0016
IOC0	0015
IOC1	0016
PWM_CONTROL	0017
banking	0000
INT_PEND1	0012
INT_MASK1	0013
WSR	0014
IOS2	0017
IOC2	0017
Ax	001A
Ah	001B
Al	001A
Bx	001C
_Bh	001D
Bl	001C
Cx	001E
Ch	001F
Cl	001E
Dx	0020
Dh	0021
Dl	0020
Ex	0022

Eh	0023
El	0022
Fx	0024
Fh	0025
Fl	0024
ZERO	0000
Rcv_In_Ptr	0026
Rcv_Out_Ptr	0028
Xmt_In_Ptr	002A
Xmt_Out_Ptr	002C
Jiffy_Clock	002E
Serial_Flags	0030
Flags	0031
Flags2	0032
Xmt_Active	0000
Xmt_Full	0001
Xmt_Empty	0002
Rcv_Full	0003
Rcv_Empty	0004
CR_Detected	0005
Timer1_Int	0000
Jiffy_Override	0001
Accel	0003
T_Full	0004
Cr	000D
Lf	000A
Bs	0008
Del	007F
Bel	0007
FF	000C
Ht	0009
Vt	000B
Sp	0020
Port1_init	0000
Port2_init	00FF
TI	0005
RI	0006

SEGMENTS IN THE MODULE
==========

ISR
Relative segment, address : Not applicable

LOAD_ISR
Relative segment, address : 4000 - 42FF

LOCALS	ADDRESS
Rom_Start	4000
CS	4000
Key	4002
Month	400A
Day	400C
Year	400E
T_Overflow	4010
AD_Complete	4020
HSI_Available	4030

HSO	4040
HSI1	4050
Trap_Vector	4060
OpCode	4070
HSI4	4080
T2cap	4090
T2overflow	40A0
ExInt_Pin	40B0
HSI_Full	40C0
NMI	40D0
Software_Timer_ISR	4100
STI_10	411C
STI_30	4122
Serial_ISR	4200
SI_1	4210
SI_2	4216
Error_Int	421A
Error_Int_1	423D
Error_Int_2	4244
Error_Int_3	424B
Error_Int_4	4252
Error_Int_5	4259
Error_Int_6	4260
Error_Int_7	4267
Error_Int_8	426E
Error_Int_9	4275
Error_Int_A	427C
Error_Int_B	4283
Error_Int_C	428A
Error_Int_D	4291
Space	42FF

****

FILE NAME : timers.obj
PROGRAM MODULE, NAME : Timers

ABSOLUTE LOCALS	ADDRESS
==========	=======
IR_BASE	001A
R8	001A
R9	001B
R0	001C
R1	001D
R2	001E
R3	001F
R4	0020
R5	0021
R6	0022
R7	0023
SP	0018
STACKPOINTER	0018
AD_RESULT_LO	0002
AD_RESULT_HI	0003
AD_COMMAND	0002
HSI_MODE	0003
HSI_TIME	0004

HSI_STATUS	0006
HSO_TIME	0004
HSO_COMMAND	0006
SBUF	0007
SBUF_RX	0007
SBUF_TX	0007
INT_MASK	0008
INT_PEND	0009
WATCHDOG	000A
TIMER1	000A
TIMER2	000C
BAUD_RATE	000E
IO_PORT0	000E
IO_PORT1	000F
IO_PORT2	0010
SP_STAT	0011
SP_CON	0011
IOS0	0015
IOS1	0016
IOC0	0015
IOC1	0016
PWM_CONTROL	0017
banking	0000
INT_PEND1	0012
INT_MASK1	0013
WSR	0014
IOS2	0017
IOC2	0017
Ax	001A
Ah	001B
Al	001A
Bx	001C
_Bh	001D
Bl	001C
Cx	001E
Ch	001F
Cl	001E
Dx	0020
Dh	0021
Dl	0020
Ex	0022
Eh	0023
El	0022
Fx	0024
Fh	0025
Fl	0024
ZERO	0000
Rcv_In_Ptr	0026
Rcv_Out_Ptr	0028
Xmt_In_Ptr	002A
Xmt_Out_Ptr	002C
Jiffy_Clock	002E
Serial_Flags	0030
Flags	0031
Flags2	0032
Xmt_Active	0000

LOADER.ASM

```
Archimedes 8096/196 Assembler V1.95/MD2          25/Oct/90  14:11:31

 Source   =   loader.asm
 List     =   loader.lst
 Object   =   loader.obj
 Options  =
                        (c) Copyright Archimedes Software Inc. 1988
```

```
;*****************************************************************************
;*                                                                           *
;*                              L O A D E R . a s m                          *
;*                                                                           *
;*                                  written by                               *
;*                              John Dombrowski                              *
;*                          University Of Michigan                           *
;*                     Space Physics Research Laboratory                     *
;*                                2455 Hayward                               *
;*                            Ann Arbor, MI  48109                           *
;*                              (313) 422-1737                               *
;*                                                                           *
;*                          October 19, 1990 anno domini                     *
;*                                                                           *
;*          This program is used to download Intel Standard Hex into         *
;*     the DATES control computer which uses the SEEQ 27HC256 EEPROM for     *
;*     non-volital program storage.                                          *
;*          Since the 80C196 microcontroller uses memory mapped              *
;*     interrupt vectors near it's code entry point this program maps        *
;*     the interrupt vectors to fixed memory locations within the EEPROM     *
;*     address space.                                                        *
;*          Upon Reset this program looks for an ID key in the EEPROM        *
;*     address space and possiblely a checksum.  If both are found           *
;*     then the EEPROM software will be executed from its entry point.       *
;*     If they are not found then the loader program will automatically      *
;*     execute.                                                              *
;*                                                                           *
;*          The Loader program will understand command lines beginning       *
;*     with a single command letter and terminating with a Carriage Return.  *
;*     Spaces are ignored. Upper Case characters only.                       *
;*                                                                           *
;*     ':'              Load a line of Intel Standard HEX                    *
;*     'R' aaaa llll    Dump hex code from start address a, length l         *
;*     'C'              Calculate EEPROM Checksum, and store it in EEPROM    *
;*     'G'              Go from beginning of EEPROM program                  *
;*     'W' aaaa dd ..   Write data('s) d starting at address a               *
;*                                                                           *
;*          The Loader operates 1/2 Duplex, 9600 baud, No Parity.            *
;*          The Loader remaps interrupts to EEPROM.                          *
;*          The Loader Prompt '>' when it is ready for the next inst.        *
;*                                                                           *
;*****************************************************************************
```

```
;       Assembler Directives

50  0000        LSTCOD  +
51  0000        LSTWID  -
52  0000        LSTPAG  +
53  0000        LSTXRF
54  0000        PAGSIZ  56
```

56
57
58

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                          A D D R E S S   E Q U A T E S                    *
                ;*                                                                           *
                ;*****************************************************************************

4300            Dates_Start     equ     4300h           ; EEPROM Progam Start Address

4010            Timer_Overflow  equ     4010h           ; Timer Overflow Vector
4020            AD_Complete     equ     4020h           ; A/D conversion Complete Vector
4030            HSI_Available   equ     4030h           ; HSI Data Available Vector
4040            HSO             equ     4040h           ; High Speed Output Vector
4050            HSI1            equ     4050h           ; HSI.1 External Interrupt Vector
4100            Sftwr_Timer     equ     4100h           ; Software Timer Interrupt
4200            Serial          equ     4200h           ; Serial Interrupt
40E0            ExInt           equ     40E0h           ; External Interrupt
4060            Trap_Vector     equ     4060h           ; Trap Interupt
4070            OpCode          equ     4070h           ; Unimplimented OpCode Interrupt

4200            Ti              equ     4200h           ; Serial Xmt Interrupt
4200            Ri              equ     4200h           ; Serial Rcv Interrupt
4080            HSI4            equ     4080h           ; 4th HSI FIFO entry Vector
4090            T2cap           equ     4090h           ; Timer2 Capture Vector
40A0            T2overflow      equ     40A0h           ; Timer2 Overflow Vector
40B0            ExInt_Pin       equ     40B0h           ; External Interrupt pin Vector
40C0            HSI_Full        equ     40C0h           ; HSI FIFO Full Vector
40D0            NMI             equ     40D0h           ; NMI Vector

4000            CS              equ     4000h           ; Checksum Word
4002            Key             equ     4002h           ; Program Loaded Key
4008            Month           equ     4008h           ; Compile Month
4009            Day             equ     4009h           ;         Day
400A            Year            equ     400Ah           ;         Year

4000            EEPROM_start    equ     4000h           ; checksum start address
C000            EEPROM_stop     equ     0C000h          ; checksum stop address

000D            Cr              equ     #0Dh            ; Carriage Return
000A            Lf              equ     #0Ah            ; Line Feed
007F            Del             equ     #7Fh            ; Delete
0008            Bs              equ     #08h            ; Back Space


0050            Key1            equ     'P'             ; Key Codes
0026            Key2            equ     '&'
0047            Key3            equ     'G'
0020            Key4            equ     ' '
0049            Key5            equ     'I'
006E            Key6            equ     'n'
0063            Key7            equ     'c'

0008            Port1_init      equ     00001000b       ; Acc Controls inactive, Cam Act Low
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                    R E G I S T E R   D E F I N A T I O N S                *
                ;*                                                                           *
                ;*****************************************************************************


                $reg8096.inc
                ;       - REG8096.INC -
                ;
                ;       Include file containing 8096 register definitions
                ;


                ;
                ;       ICC Working registers
                ;
 001A           IR_BASE DEFINE  01AH            ; Register base address

 001A           R8      DEFINE  (IR_BASE)
 001B           R9      DEFINE  (IR_BASE+1)

 001C           R0      DEFINE  (IR_BASE+2)
 001D           R1      DEFINE  (IR_BASE+3)
 001E           R2      DEFINE  (IR_BASE+4)
 001F           R3      DEFINE  (IR_BASE+5)
 0020           R4      DEFINE  (IR_BASE+6)
 0021           R5      DEFINE  (IR_BASE+7)
 0022           R6      DEFINE  (IR_BASE+8)
 0023           R7      DEFINE  (IR_BASE+9)

                ;
                ;       Special Function Registers
                ;
 0018           SP              DEFINE  18H
 0018           STACKPOINTER    DEFINE  18H

 0002           AD_RESULT_LO    DEFINE  02H
 0003           AD_RESULT_HI    DEFINE  03H
 0002           AD_COMMAND      DEFINE  02H

 0003           HSI_MODE        DEFINE  03H
 0004           HSI_TIME        DEFINE  04H
 0006           HSI_STATUS      DEFINE  06H

 0004           HSO_TIME        DEFINE  04H
 0006           HSO_COMMAND     DEFINE  06H

 0007           SBUF            DEFINE  07H
 0007           SBUF_RX         DEFINE  07H
 0007           SBUF_TX         DEFINE  07H

 0008           INT_MASK        DEFINE  08H
```

```
0009        INT_PENDING     DEFINE  09H

000A        WATCHDOG        DEFINE  0AH

000A        TIMER1          DEFINE  0AH
000C        TIMER2          DEFINE  0CH

000E        BAUD_RATE       DEFINE  0EH

000E        IO_PORT0        DEFINE  0EH
000F        IO_PORT1        DEFINE  0FH
0010        IO_PORT2        DEFINE  10H

0011        SP_STAT         DEFINE  11H
0011        SP_CON          DEFINE  11H

0015        IOS0            DEFINE  15H
0016        IOS1            DEFINE  16H

0015        IOC0            DEFINE  15H
0016        IOC1            DEFINE  16H

0017        PWM_CONTROL     DEFINE  17H

0000        ZERO            DEFINE  00h
0000        ZEROh           DEFINE  00h
0001        ZERO1           DEFINE  01h
                    ; 80c196 Special Function Registers

0012        IPEND1          DEFINE  12h
0013        IMASK1          DEFINE  13h
0014        WSR             DEFINE  14h
0017        IOS2            DEFINE  17h
0017        IOC2            DEFINE  17h

001A        Ax              Define          1Ah
001B        Ah              Define          1Bh
001A        Al              Define          1Ah

001C        Bx              Define          1Ch
001C        Bl              Define          1Ch

001E        Cx              Define          1Eh
001E        Cl              Define          1Eh
001F        Ch              Define          1Fh

0020        Rcv_Ptr         Define          20h             ; Word
0022        Length          Define          22h             ; Byte
0023        Type            Define          23h             ; Byte
0024        Address         Define          24h             ; Word
0026        Check_Sum       Define          26h             ; Byte
0027        CS_Calc         Define          27h             ; Byte
0028        Down_Load_Err   Define          28h             ; Word
```

220

```
222
223                 ;*****************************************************************************
224                 ;*                                                                           *
225                 ;*                          I N T E R N A L   R A M                          *
226                 ;*                                                                           *
227                 ;*****************************************************************************
228
229  0000                   RSEG    IntRam
230
231  0000           Ram_Start:
232
233  0018                   org Ram_Start + 18h
234
235  0018                           DSB     2       ; SP
236  001A                           DSB     2       ; Ax
237  001C                           DSB     2       ; Bx
238  001E                           DSB     2       ; Cx
239  0020                           DSB     2       ; Rcv_Ptr
240  0022                           DSB     1       ; Length
241  0023                           DSB     1       ; Type
242  0024                           DSB     2       ; Address
243  0026                           DSB     1       ; CheckSum
244  0027                           DSB     1       ; CS_Calc
245  0028                           DSB     2       ; Down_Load_Err
246
247  002A           Hex_Data:       DSB     16      ; Intel Standard Hex Data Buffer
248  003A           Rcv_Buff:       DSB     60      ; Serial Recieve Buffer
249  0076           Rcv_End:
250
251  0000                   RSEG    Stack
252
253  0000                           DSB     100     ; Stack Buffer
254  0064           Stack_Ptr:
255
```

```
25
258                 ;*****************************************************************************
259                 ;*                                                                           *
260                 ;*                          I N T E R R U P T   V E C T O R S                *
261                 ;*                                                                           *
262                 ;*****************************************************************************
263
264  0000                   RSEG    ROM (2)
265  0000           RomStart:
266
267                                 ; 8096 Interrupts
268
269  0000 1040              DCW     Timer_Overflow          ; 4010h
270  0002 2040              DCW     AD_Complete             ; 4020h
271  0004 3040              DCW     HSI_Available           ; 4030h
272  0006 4040              DCW     HSO                     ; 4040h
273  0008 5040              DCW     HSI1                    ; 4050h
274  000A 0041              DCW     Sftwr_Timer             ; 4100h
275  000C 0042              DCW     Serial                  ; 4200h
276  000E E040              DCW     ExInt                   ; 40E0h
277  0010 6040              DCW     Trap_Vector             ; 4060h
278  0012 7040              DCW     OpCode                  ; 4070h
279
280                                 ; 80c196 Interrupts
281
282  0030                   ORG     RomStart + 30h
283
284  0030 0042              DCW     Serial                  ; 4200h   Ti
285  0032 0042              DCW     Serial                  ; 4200h   Ri
286  0034 8040              DCW     HSI4                    ; 4080h   4th HSI FIFO entry
287  0036 9040              DCW     T2cap                   ; 4090h   Timer 2 capture
288  0038 A040              DCW     T2overflow              ; 40A0h   Timer 2 overflow
289  003A B040              DCW     ExInt_Pin               ; 40B0h   External Interrupt Pin
290  003C C040              DCW     HSI_Full                ; 40C0h   HSI FIFO Full
291  003E D040              DCW     NMI                     ; 40D0h   Non Maskable Interrupt
292
293  0018                   org     RomStart + 18h
294
295  0018 CB        CCR:    DCB     11001011b               ; Enable Power Down
296                                                        ; 16 bit Bus Width
297                                                        ; /Wrl & /Wrh write strobes
298                                                        ; ALE
299                                                        ; Ready = 1 wait state
300                                                        ; No Program Lock
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                              R E S E T                                    *
                ;*                                                                           *
                ;*****************************************************************************

                ;       Reset Entry Point



0080                    org     RomStart + 80h

0080 A102401C   Reset:  ld      Bx, #Key                        ; Bx = Key Address
0084 B21D1A             ldb     Al, [Bx]+                       ; read Key1
0087 99501A             cmpb    Al, #Key1                       ; test for key 1 match
008A D736               jne     No_Key_Match

008C B21D1A             ldb     Al, [Bx]+                       ; Read Key2
008F 99261A             cmpb    Al, #Key2
0092 D72E               jne     No_Key_Match

0094 B21D1A             ldb     Al, [Bx]+                       ; Read Key2
0097 99471A             cmpb    Al, #Key3
009A D726               jne     No_Key_Match

009C B21D1A             ldb     Al, [Bx]+                       ; Read Key2
009F 99201A             cmpb    Al, #Key4
00A2 D71E               jne     No_Key_Match

00A4 B21D1A             ldb     Al, [Bx]+                       ; Read Key2
00A7 99491A             cmpb    Al, #Key5
00AA D716               jne     No_Key_Match

00AC B21D1A             ldb     Al, [Bx]+                       ; Read Key2
00AF 996E1A             cmpb    Al, #Key6
00B2 D70E               jne     No_Key_Match

00B4 B21D1A             ldb     Al, [Bx]+                       ; Read Key2
00B7 99631A             cmpb    Al, #Key7
00BA D706               jne     No_Key_Match

00BC A100431C           ld      Bx, #Dates_Start
00C0 E31C               br      [Bx]                            ; all Keys match. Run Dates

00C2            No_Key_Match:
00C2 E73B00             br      Loader_Start                    ; Key Mis-Match. Run Loader
```

```
35
351                 ;*****************************************************************************
352                 ;*                                                                           *
353                 ;*                    L O A D E R   E N T R Y   P O I N T                    *
354                 ;*                                                                           *
355                 ;*****************************************************************************
356
357  0100                   org     RomStart + 100h                  ; Loader Address = 2100h
358
359  0100           Loader_Start:
360  0100 A1640018          ld      SP, #Stack_Ptr                   ; Set stack pointer
361  0104 1114              clrb    WSR                              ; 80c196 Window Register
362  0106 FA                di                                       ; disable all interrupts
363  0107 B15503            ldb     HSI_MODE, #01010101b
364  010A B1080F            ldb     IO_PORT1, #Port1_init            ; init Port1
365
366                         ; Initialize Serial Port
367
368  010D B1A015            ldb     IOC0, #10100000b                 ; HSI.0 disabled
369                                                                 ; T2 not reset each write
370                                                                 ; HSI.1 disabled
371                                                                 ; T2 ext rst disabled
372                                                                 ; HSI.2 disabled
373                                                                 ; T2 rst = HSI.0
374                                                                 ; HSI.3 disabled
375                                                                 ; T2 clk source = HSI.1
376  0110 B1A116            ldb     IOC1, #10100001b                 ; PWM enabled
377                                                                 ; ACH.7 ext intr disabled
378                                                                 ; T1 ovflo intr disabled
379                                                                 ; T2 ovflo intr disabled
380                                                                 ; HSO.4 disabled
381                                                                 ; TXD enabled
382                                                                 ; HSO.5 disabled
383                                                                 ; HSI intr = FIFO full
384  0113 B18017            ldb     IOC2, #10000000b                 ; Clear CAM = YES
385                                                                 ; Cam Lock = NO
386                                                                 ; T2 alt intr = NO
387                                                                 ; A/D prescaler = Normal
388                                                                 ; A/D sample hold = Enabled
389                                                                 ; Slow PWM = NO
390                                                                 ; T2 U/D ena = NO
391                                                                 ; Fast T2 ena = NO
392
393  0116 1108              clrb    INT_MASK
394  0118 1113              clrb    IMASK1
395  011A 1109              clrb    INT_PENDING
396  011C 1112              clrb    IPEND1
397  011E B10117            ldb     PWM_CONTROL, #1
398
399  0121 B14D0E            ldb     BAUD_RATE, # Low  804Dh          ; Baud Rate = 9600 Baud
400  0124 B1800E            ldb     BAUD_RATE, # High 804Dh          ;      xtal = clock source
401  0127 B10911            ldb     SP_CON, #00001001b               ; No Parity
402                                                                 ; Receiver Enabled
```

```
403
404
405  012A A138041A        ld      Ax, #Msg1        ; Serial Port Mode 1
406  012E EFF302          call    Send_Msg         ; Send Loader Message
407
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                              M A I N   L O O P                            *
                ;*                                                                           *
                ;*****************************************************************************

0131            New_Line:
0131 B13E07             ldb     SBUF, #'>'              ; send ready prompt
0134 A13A0020           ld      Rcv_Ptr, #Rcv_Buff      ; set Rcv ptr to top of buffer

0138 91800F     Loop:   orb     IO_PORT1, #80h          ; Pulse WatchDog Timer
013B 717F0F             andb    IO_PORT1, #7Fh

013E B0111A             ldb     Al, SP_STAT             ; read serial port status
0141 361AF4             jbc     Al, 6, Loop             ; jump if no char rcv'd

0144 B0071A             ldb     Al, SBUF                ; Read character
0147 99031A             cmpb    Al, #03                 ; check for ^C
014A D701               jne     L1
014C FF                 rst                             ; ReBoot on reciept of ^C

014D 99201A     L1:     cmpb    Al, #' '                ; ignore spaces
0150 DFE6               je      Loop

0152 990A1A             cmpb    Al, #Lf                 ; ignore line feeds
0155 DFE1               je      Loop

0157 99081A             cmpb    Al, #Bs                 ; check for Back Space
015A D704               jne     L2
015C 0520               dec     Rcv_Ptr                 ; remove one character from Rcv Buffer
015E 27D8               sjmp    Loop

0160 C6211A     L2:     stb     Al, [Rcv_Ptr]+          ; stuff it into Rcv_Buffer
0163 990D1A             cmpb    Al, #Cr                 ; Test for Carriage Return
0166 D7D0               jne     Loop



0168 B3013A00   Test1:  ldb     Al, Rcv_Buff            ; Recall first character from Buffer
016C 1A
016D 993A1A             cmpb    Al, #':'                ; Test for Intel Standard Hex
0170 D705               jne     Test2
0172 EF7801             call    Intel_Standard_Hex
0175 27BA               sjmp    New_Line

0177 993F1A     Test2:  cmpb    Al, #'?'                ; Test for Help
017A D709               jne     Test3
017C A180041A           ld      Ax, #Msg3
0180 EFA102             call    Send_Msg
0183 27AC               sjmp    New_Line

0185 990D1A     Test3:  cmpb    Al, #Cr
```

```
0188 D702                jne     Test4
018A 27A5                sjmp    New_Line                ; ignore null lines w/ only Cr

018C 99521A     Test4:   cmpb    Al, #'R'                ; Test for Read Command
018F D705                jne     Test5
0191 EFD600              call    Read
0194 279B                sjmp    New_Line

0196 99571A     Test5:   cmpb    Al, #'W'                ; Test for write command
0199 D705                jne     Test6
019B EF6500              call    Write
019E 2791                sjmp    New_Line

01A0 99431A     Test6:   cmpb    Al, #'C'                ; Test for checksum command
01A3 D705                jne     Test7
01A5 EF1700              call    CheckSum
01A8 2787                sjmp    New_Line

01AA 99471A     Test7:   cmpb    Al, #'G'                ; Test for Go Command
01AD D707                jne     Test8
01AF A100431A            ld      Ax, #Dates_Start        ; exit loader. Vector to EEPROM start
01B3 C81A                push    Ax                      ;       address.
01B5 F0                  ret

01B6 A168041A   Test8:   ld      Ax, #Msg2               ; Send Invalid response msg
01BA EF6702              call    Send_Msg
01BD 2772                sjmp    New_Line
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                              C H E C K   S U M                            *
                ;*                                                                           *
                ;*****************************************************************************

                ;               This routine calculates the checksum of the EEPROM and stores
                ;       it in the EEPROM address reserved for it.

01BF            CheckSum:
01BF 0128               clr     Down_Load_Err
01C1 A100401C           ld      Bx, #EEPROM_start
01C5 011A               clr     Ax

01C7 661D1A     CS_10:  add     Ax, [Bx]+
01CA 8900C01C           cmp     Bx, #EEPROM_stop
01CE D7F7               jne     CS_10

01D0 C81A               push    Ax
01D2 A1004024           ld      Address, #CS
01D6 EFDD01             call    Write_Memory                    ; store CheckSum in EEPROM
01D9 B01B1A             ldb     Al, Ah
01DC EFD701             call    Write_Memory
01DF CC1A               pop     Ax

01E1 A01A1C             ld      Bx, Ax                          ; save checksum in Bx

01E4 A1D9061A           ld      Ax, #Msg13                      ; send checksum result message
01E8 D303               jnc     CS_20                           ; jump if no error occured in
01EA A0281A             ld      Ax, Down_Load_Err               ;   Memory Write

01ED EF3402     CS_20:  call    Send_Msg
01F0 B01D1A             ldb     Al, (Bx+1)
01F3 EF0602             call    Send_Byte
01F6 B01C1A             ldb     Al, Bx
01F9 EF0002             call    Send_Byte
01FC B10D1A             ldb     Al, #Cr
01FF EF1802             call    Send_Char
0202 F0                 ret
```

```
                ;*****************************************************************************
                ;*                                W R I T E                                  *
                ;*                                                                           *
                ;*****************************************************************************

                ;               This routine reads a 4 digit HEX address and a variable number
                ;       of 2 digit HEX data values and writes the data sequentally
                ;       starting from the specified address.

                ;               The address must be four characters long.

                ;               Each data byte must be 2 characters long, there may be
                ;       up to 25 data bytes per input line.  Spaces are ignored.


0203 0128       Write:  clr     Down_Load_Err
0205 0124               clr     Address
0207 A13B0020           ld      Rcv_Ptr, #Rcv_Buff + 1
                                                        ; read 4 char address
                        ldb     Cl, #4                  ; read a character from Rcv Buffer
020B B1041E     W_10:   ldb     Ah, [Rcv_Ptr]+          ; Test for end of buffer
020E B2211B             cmpb    Ah, #Cr                 ; jump if error in address length
0211 990D1B             je      W_Error1
0214 DF40                                               ; shift Address up one nibble
                        shl     Address, #4             ; conv ASCII char to HEX
0216 090424             call    Conv_Asc_Hex
0219 EF7E01             jc      W_Error2                ; combine HEX data w/ Address
021C DB3E               orb     Address, Ah             ; loop until address construction done
021E 901B24             dbnz    Cl, W_10
0221 E01EEA                                             ; Clear data storage variable
                W_20:   clrb    Al                      ; Read High Data Nibble Rcv Buffer
0224 111A               ldb     Ah, [Rcv_Ptr]+          ; Test if Done
0226 B2211B             cmpb    Ah, #Cr
0229 990D1B             jne     W_30                    ; exit if buffer is empty
022C D701               ret                             ; conv ASCII char to HEX
022E F0         W_30:   call    Conv_Asc_Hex            ; exit on conversion error
022F EF6801             jc      W_Error2                ; swap HEX to upper nibble
0232 DB28               shlb    Ah, #4                  ; store HEX in Al
0234 19041B             ldb     Al, Ah                  ; read lower data nibble Rcv Buffer
0237 B01B1A             ldb     Ah, [Rcv_Ptr]+          ; test for empty buffer
023A B2211B             cmpb    Ah, #13                 ; Exit on Error
023D 990D1B             je      W_Error3                ; conv ASCII char to HEX
0240 DF20               call    Conv_Asc_Hex            ; Exit on conversion error
0242 EF5501             jc      W_Error2                ; combine upper and lower nibbles
0245 DB15               orb     Al, Ah                  ; write data to memory
0247 901B1A             call    Write_Memory            ; exit if an error occurs
024A EF6901             jc      W_Error0                ; read next data value from Rcv Buff
024D DB02               sjmp    W_20
024F 27D3
                W_Error0:
0251                    ld      Ax, Down_Load_Err
0251 A0281A
```

```
0254 2010                 sjmp    W_40

0256                W_Error1:
0256 A163061A             ld      Ax, #Msg10          ; Address Length Error Message
025A 200A                 sjmp    W_40

025C                W_Error2:
025C A18D061A             ld      Ax, #Msg11          ; Non Ascii character error message
0260 2004                 sjmp    W_40

0262                W_Error3:
0262 A1B2061A             ld      Ax, #Msg12          ; Data Length Error

0266 EFBB01         W_40:   call    Send_Msg            ; send error message
0269 F0                     ret                         ; Exit
```

```
608                 ;*****************************************************************************
609                 ;*                                                                           *
610                 ;*                                  R E A D                                  *
611                 ;*                                                                           *
612                 ;*****************************************************************************
613
614                 ;       This routine takes an address and an optional length and dumps data
615                 ;       from memory to the serial port. Addres and length are specified in HEX.
616                 ;
617                 ;               The Address may be up to 4 characters in length if checking
618                 ;       a single byte.
619                 ;
620                 ;               The Address must be 4 characters long if the length is being
621                 ;       specified.  The Length may be up to 4 characters long.
622                 ;
623
624
625  026A           Read:
626  026A 0128              clr     Down_Load_Err                   ; init variables
627  026C 0124              clr     Address
628  026E 0122              clr     Length
629  0270 A13B0020          ld      Rcv_Ptr, #Rcv_Buff + 1          ; set ptr past 'd' command
630
631  0274 B1041E            ldb     Cl, #4                          ; read up to 4 char into Address
632  0277 B2211B    D_10:   ldb     Ah, [Rcv_Ptr]+                  ; read ASCII Character
633  027A 990D1B            cmpb    Ah, #Cr                         ; test for end of Rcv Buff
634  027D DF27              je      D_30                            ;
635
636  027F 090424            shl     Address, #4                     ; shift Addr up 1 nibble
637  0282 EF1501            call    Conv_Asc_Hex                    ; ASCII char >>> Hex number
638  0285 DB5E              jc      D_Error                         ; Exit if Non Ascii char
639  0287 901B24            orb     Address, Ah                     ; Else, combine Hex W/ Addr
640  028A E01EEA            dbnz    Cl, D_10                        ; read next address char
641
642  028D B1041E            ldb     Cl, #4                          ; Read up to 4 Length char
643  0290 B2211B    D_20:   ldb     Ah, [Rcv_Ptr]+                  ; Read character from Rcv Buff
644  0293 990D1B            cmpb    Ah, #Cr                         ; test for end of Rcv Buffer
645  0296 DF0E              je      D_30
646
647  0298 090422            shl     Length, #4                      ; shift length up 1 nibble
648  029B EFFC00            call    Conv_Asc_Hex                    ; ASCII char >>> Hex Number
649  029E DB45              jc      D_Error                         ; Exit if Non ASCII char
650  02A0 901B22            orb     Length, Ah                      ; Combine Hex number w/ Length
651  02A3 E01EEA            dbnz    Cl, D_20                        ; Read next Length Character
652
653  02A6 B0251A    D_30:   ldb     Al, Address + 1                 ; Send Dump Address
654  02A9 EF5001            call    Send_Byte
655  02AC B0241A            ldb     Al, Address
656  02AF EF4A01            call    Send_Byte                       ; send delimiter
657  02B2 B1201A            ldb     Al, #' '
658  02B5 EF6201            call    Send_Char
659
660  02B8 89000022          cmp     Length, #0                      ; Min Length = 1
```

```
66    02BC D702                 jne     D_40
662   02BE 0722                 inc     Length
663
664   02C0 B1101E     D_40:     ldb     Cl, #16                  ; dump 16 numbers / line
665   02C3 B2251A     D_50:     ldb     Al, [Address]+           ; read data from memory
666   02C6 EF3301               call    Send_Byte                ; and send it
667   02C9 B1201A               ldb     Al, #' '                 ; Send space delimiter
668   02CC EF4B01               call    Send_Char
669   02CF 0522                 dec     Length                   ; bump xmt length counter
670   02D1 DF0B                 je      D_Done                   ; quit when length = 0
671   02D3 E01EED               dbnz    Cl, D_50                 ; finish dump line
672   02D6 B10D1A               ldb     Al, #Cr                  ; terminate dump line w/ Cr
673   02D9 EF3E01               call    Send_Char
674   02DC 27C8                 sjmp    D_30                     ; Begin New Dump Line
675
676   02DE B10D1A     D_Done:   ldb     Al, #Cr                  ; end transmission w/ final Cr
677   02E1 EF3601               call    Send_Char
678   02E4 F0                   ret
679
680   02E5            D_Error:                                   ; Send Error Message
681   02E5 A13F061A             ld      Ax, #Msg9
682   02E9 EF3801               call    Send_Msg
683   02EC F0                   ret
684
685
686
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                          I N T E L   S T A N D A R D   H E X              *
                ;*                                                                           *
                ;*****************************************************************************


02ED            Intel_Standard_Hex:
                        clr     Down_Load_Err           ; clear transmission error variable
02ED 0128               clrb    CS_Calc                 ; clear check cum calculation
02EF 1127

                        ld      Bx, #Rcv_Buff + 1       ; set Bx to Rcv Buffer, past ':' char
02F1 A13B001C           call    Asc_Hex                 ; Read Length character
02F5 EF8900             ldb     Length, Al
02F8 B01A22             call    Asc_Hex                 ; Read Upper byte of address
02FB EF8300             ldb     (Address + 1), Al
02FE B01A25             call    Asc_Hex                 ; Real Lower byte of address
0301 EF7D00             ldb     (Address + 0), Al
0304 B01A24             call    Asc_Hex                 ; Read Type
0307 EF7700             ldb     Type, Al
030A B01A23

                        cmp     Bx, Rcv_Ptr             ; check if Rcv_Buff is empty
030D 88201C             jgt     ISH_Length_Err          ; *** TEST MODE ***
0310 D264

                        cmpb    Length, #16             ; Test for Record Length Too Long
0312 991022             jgt     ISH_Length_Err
0315 D25F

                        push    Length                  ; save length on stack
0317 C822               ld      Cx, #Hex_Data           ; Cx points to Hex Data Buffer
0319 A12A001E           cmpb    Length, #0              ; ck if no data to convert
031D 990022             je      ISH_20
0320 DF09               call    Asc_Hex                 ; Read data byte
0322 EF5C00     ISH_10: stb     Al, [Cx]+               ; store into HEX data buffer
0325 C61F1A             dbnz    Length, ISH_10
0328 E022F7

                ISH_20: pop     Length                  ; Restore Length from stack
032B CC22               call    Asc_Hex                 ; Read checksum
032D EF5100             ldb     Check_Sum, Al
0330 B01A26

                        cmp     Bx, Rcv_Ptr             ; check if Rcv_Buff contains less
0333 88201C             jgt     ISH_Length_Err          ;     characters than were required
0336 D23E

                        cmpb    CS_Calc, #0             ; check for checksum error
0338 990027             jne     ISH_CS_Err
033B D733

                        cmp     Down_Load_Err, #0       ; check if Asc_Hex routine detected
033D 89000028           jne     ISH_Error               ;     any download errors
0341 D737


                        cmpb    Type, #00               ; test for Hex Record
0343 990023             je      ISH_Hex_Record
0346 DF06
```

```
0348 990123          cmpb    Type, #01              ; test for end of load record
034B D717            jne     ISH_Type_Err           ; exit on end of load.
034D F0              ret

                                  ; Write data to EEPROM

034E            ISH_Hex_Record:
034E A12A001C        ld      Bx, #Hex_Data          ; Make Sure length <> 0
0352 990022          cmpb    Length, #00
0355 D701            jne     ISHH_20
0357 F0              ret

0358 B21D1A     ISHH_20:ldb  Al, [Bx]+              ; Read Data Byte
035B EF5800          call    Write_Memory           ; Write to memory
035E DB1A            jc      ISH_Error              ;   Exit if EEPROM error occurs
0360 E022F5          dbnz    Length, ISHH_20        ; Repeat until all characters are written
0363 F0              ret                            ; Exit

0364            ISH_Type_Err:
0364 A1100628        ld      Down_Load_Err, #Msg8
0368 2010            sjmp    ISH_Error

036A            ISH_Range_Error:
036A A1DE0528        ld      Down_Load_Err, #Msg7
036E 200A            sjmp    ISH_Error

0370            ISH_CS_Err:
0370 A1B90528        ld      Down_Load_Err, #Msg6   ; Checksum Error message
0374 2004            sjmp    ISH_Error

0376            ISH_Length_Err:
0376 A18C0528        ld      Down_Load_Err, #Msg5   ; Incorrect Record length message

037A            ISH_Error:
037A A0281A          ld      Ax, Down_Load_Err
037D EFA400          call    Send_Msg
0380 F0              ret

                ;*****************************************************************************
                ;*                                                                           *
                ;*                              A S C   H E X                                *
                ;*                                                                           *
                ;*****************************************************************************

                ;               This routine reads two ASCII characters in the Rcv Buffer
                ;       pointed by Bx and converts them into a HEX number.  The result is
                ;       Passed back in Al.

0381            Asc_Hex:
0381 B21D1B          ldb     Ah, [Bx]+              ; read first character
```

```
0384 EF1300          call    Conv_Asc_Hex        ; shift Hex to Upper Nibble
0387 19041B          shlb    Ah, #4              ; store in Al
038A B01B1A          ldb     Al, Ah              ; read second character
038D B21D1B          ldb     Ah, [Bx]+
0390 EF0700          call    Conv_Asc_Hex        ; combine low nibble w/ high nibble
0393 901B1A          orb     Al, Ah
                                                 ; calc checksum
0396 741A27          addb    CS_Calc, Al
0399 F0              ret


039A                 Conv_Asc_Hex:
039A 79301B                  subb    Ah, #'0'    ; subtract ASCII '0'
039D D311                    jnc     CAH_10      ; non ASCII on borrow. C = /borrow
039F 990A1B                  cmpb    Ah, #10
03A2 DE0A                    jlt     CAH_20
03A4 79071B                  subb    Ah, #7      ; jump no borrow, Non ASCII
03A7 D307                    jnc     CAH_10
03A9 990F1B                  cmpb    Ah, #0Fh
03AC D202                    jgt     CAH_10      ; clear carry on good conversion
03AE F8              CAH_20: clrc
03AF F0                      ret
                                                 ; Non ASCII Character
03B0 A1630528        CAH_10: ld      Down_Load_Err, #Msg4   ; set carry on error condition
03B4 F9                      setc
03B5 F0                      ret
```

```
827                 ;*****************************************************************
828                 ;*                          W R I T E   M E M O R Y              *
829                 ;*                                                               *
830                 ;*****************************************************************
831                 ;
832                 ;       This routine recieves a data byte in Al and its address in
833                 ;       ADDRESS.  If the address of the data byte is in the range of the
834                 ;       EEPROM then this routine will use a /DATA delay algorythum while
835                 ;       the EEPROM performs its write cycle.  Otherwise a normal memory write
836                 ;       will be performed.  The EEPROM write cycle delay will not extend beyond
837                 ;       20 mSec.
838                 ;               After the data has been written it will be read back and XOR'd
839                 ;       with the origonal data.  If there is an error then Down_Load_Err
840                 ;       variable will be set to Error Message #14.
841                 ;               This routine does not alter any registers.
842                 ;               Address is incremented by this routine.
843                 ;               Carry set if a write error occurs, otherwise it is cleared
844                 ;
845
846
847  03B6           Write_Memory:
848                                 push    Ax
849  03B6 C81A                      push    Bx                      ; write data into memory
850  03B8 C81C                      stb     Al, [Address]
851  03BA C6241A                                                    ; check address range
852
853                                 sub     Bx, Address, #EEPROM_start
854  03BD 49004024                                                  ; jump if Address below EEPROM
     03C1 1C                        jnc     Verify_Write
855  03C2 D326                      sub     Bx, Address, #EEPROM_stop
856  03C4 4900C024                                                  ; jump if Address above EEPROM
     03C8 1C                        jc      Verify_Write
857  03C9 DB1F                      ; EEPROM /DATA delay
858
859                 EEPROM_delay:                                  ; wait up to 20 mSec for EEPROM to
860  03CB                           ldb     Ah, #20                ;    complete write cycle
861  03CB B1141B                                                   ; load Bx w/ 1 mSec delay constant
862                 ED_00:          ld      Bx, #375
863  03CE A177011C  ED_10:          dec     Bx
864  03D2 051C                      jne     ED_10
865  03D4 D7FC                                                     ; check if EEPROM has completed write
866                                 cmpb    Al, [Address]
867  03D6 9A241A                    je      Verify_Write
868  03D9 DF0F                      dbnz    Ah, ED_00
869  03DB E01BF0                                                   ; EEPROM Delay Write Failure
870                                 ld      Down_Load_Err, #Msg15  ; bump address
871  03DE A1010728                  inc     Address
872  03E2 0724                      pop     Bx
873  03E4 CC1C                      pop     Ax                     ; set carry on error
874  03E6 CC1A                      setc
875  03E8 F9                        ret
876  03E9 F0
877
```

```
03EA                Verify_Write:
03EA B2251B                 ldb     Ah, [Address]+          ; readback address, bump address
03ED 941B1A                 xorb    Al, Ah
03F0 DF05                   je      VW_10                   ; jump if write verifys
03F2 A1EC0628               ld      Down_Load_Err, #Msg14   ; else send Down load error message
03F6 F9                     setc                            ;      and set carry

03F7 CC1C           VW_10:  pop     Bx
03F9 CC1A                   pop     Ax
03FB F0                     ret
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                          S E N D    B Y T E                   *
                ;*                                                               *
                ;*****************************************************************

                ;               This routine sends the byte in Al as a two character
                ;       ASCII encoded HEX value.

                Send_Byte:
03FC                    push    Ax              ; save character
03FC C81A               andb    Al, #0F0h       ; mask off upper nibble
03FE 71F01A             shrb    Al, #4          ; move upper nibble to lower nibble
0401 18041A             scall   SB_10           ; send upper nibble
0404 2805               pop     Ax              ; restore character
0406 CC1A               andb    Al, #0Fh        ; mask of upper nibble
0408 710F1A                                     ; then send upper nibble

                SB_10:  addb    Al, #'0'        ; convert Hex Character to ASCII
040B 75301A             cmpb    Al, #'9'        ; check for A-F character
040E 99391A             jle     SB_20
0411 DA03               addb    Al, #7          ; correct 0-F character
0413 75071A             call    Send_Char
0416 EF0100     SB_20:  ret
0419 F0
```

```
                ;*****************************************************************
                ;*                          S E N D     C H A R                  *
                ;*                                                               *
                ;*****************************************************************
                ;
                ;          This routine transmitts the character in A1 then
                ;       waits until the serial port is not transmitting
                ;
                ;
                Send_Char:
                        ldb     SBUF, A1                ; Read serail port status
041A            041A B01A07     SC_10:  ldb     A1, SP_STAT             ; Wait until Ti flag is set
041D B0111A             jbc     A1, 5, SC_10
0420 351AFA             ret
0423 F0
```

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                              S E N D     M S G                            *
                ;*                                                                           *
                ;*****************************************************************************

                ;                 This routine sends the ASCII string message pointed to by Ax.
                ;        Characters in the string are transmitted until a null terminator is
                ;        detected.

0424 B21B1E     Send_Msg:       ldb     Cl, [Ax]+               ; read character from table
0427 99001E                     cmpb    Cl, #00                 ; exit when a null is detected
042A DF0B                       je      SM_20

042C B01E07                     ldb     SBUF, Cl                ; Send Character
042F B0111E     SM_10:          ldb     Cl, SP_STAT             ; Wait until serial Port empty
0432 351EFA                     jbc     Cl, 5, SM_10            ;
0435 27ED                       sjmp    Send_Msg                ; Send next character

0437 F0         SM_20:          ret
```

```
961                 ;*****************************************************************
962                 ;*                          M E S S A G E   T A B L E            *
963                 ;*                                                               *
964                 ;*****************************************************************
965
966
967
968
969  0438 0D202020  Msg1:   dcb    Cr, '   *** DATES ***', Cr
     043C 2A2A2A20
     0440 44415445
     0444 53202A2A
     0448 2A0D
970  044A 45455052          dcb    'EEPROM loader Program', Cr, Lf
     044E 4F4D206C
     0452 6F616465
     0456 72205072
     045A 6F677261
     045E 6D0D0A
971  0461 52656164          dcb    'Ready', Cr, 00
     0465 790D00
972
973  0468 4D657373  Msg2:   dcb    'Message Not Understood', Cr, 00
     046C 61676520
     0470 4E6F7420
     0474 556E6465
     0478 7273746F
     047C 6F640D00
974
975  0480 3A202020  Msg3:   dcb    ':              - Download Intel Standard Hex', Cr
     0484 20202020
     0488 20202020
     048C 202D2044
     0490 6F776D6C
     0494 6F616420
     0498 496E7465
     049C 6C205374
     04A0 616E6461
     04A4 72642048
     04A8 65780D
976  04AB 47202020          dcb    'G              - Go EEPROM program', Cr
     04AF 20202020
     04B3 20202020
     04B7 202D2047
     04BB 6F204545
     04BF 50524F4D
     04C3 2070726F
     04C7 6772616D
     04CB 0D
977  04CC 43202020          dcb    'C              - Calculage Checksum - Store in EEPROM', Cr
     04D0 20202020
     04D4 20202020
     04D8 202D2043
```

```
     04DC 616C6375
     04E0 6C616765
     04E4 20436865
     04E8 636B7375
     04EC 6D202D20
     04F0 53746F72
     04F4 6520696E
     04F8 20454550
     04FC 524F4D0D
978  0500 52206161          dcb     'R aaaa 1111  - Read from {aaaa}, length {1111}', Cr
     0504 6161206C
     0508 6C6C6C20
     050C 202D2052
     0510 65616420
     0514 66726F6D
     0518 207B6161
     051C 61617D2C
     0520 206C656E
     0524 67746820
     0528 7B6C6C6C
     052C 6C7D0D
979  052F 57206161          dcb     'W aaaa dd .. - Write to {aaaa} data {dd dd .. dd}', Cr, Lf, 00
     0533 61612064
     0537 64202E2E
     053B 202D2057
     053F 72697465
     0543 20746F20
     0547 7B616161
     054B 617D2064
     054F 61746120
     0553 7B646420
     0557 6464202E
     055B 2E206464
     055F 7D0D0A00
980
981  0563 446F776E  Msg4:   dcb     'Down Load Error ... Non ASCII character', Cr, 00
     0567 204C6F61
     056B 64204572
     056F 726F7220
     0573 2E2E2E20
     0577 4E6F6E20
     057B 41534349
     057F 49206368
     0583 61726163
     0587 7465720D
     058B 00
982  058C 446F776E  Msg5:   dcb     'Down Load Error ... Incorrect Record Length', Cr, 00
     0590 204C6F61
     0594 64204572
     0598 726F7220
     059C 2E2E2E20
     05A0 496E636F
     05A4 72726563
```

```
     05A8 74205265
     05AC 636F7264
     05B0 204C656E
     05B4 6774680D
     05B8 00
983  05B9 446F776E  Msg6:   dcb     'Down Load Error ... Check Sum Error', Cr, 00
     05BD 204C6F61
     05C1 64204572
     05C5 726F7220
     05C9 2E2E2E20
     05CD 43686563
     05D1 6B205375
     05D5 6D204572
     05D9 726F720D
     05DD 00
984  05DE 446F776E  Msg7:   dcb     'Down Load Error ... Outside EEPROM address range', Cr, 00
     05E2 204C6F61
     05E6 64204572
     05EA 726F7220
     05EE 2E2E2E20
     05F2 4F757473
     05F6 69646520
     05FA 45455052
     05FE 4F4D2061
     0602 64647265
     0606 73732072
     060A 616E6765
     060E 0D00
985  0610 446F776E  Msg8:   dcb     'Down Load Error ... Unknown Intel Record Type', Cr, 00
     0614 204C6F61
     0618 64204572
     061C 726F7220
     0620 2E2E2E20
     0624 556E6B6E
     0628 6F776E20
     062C 496E7465
     0630 6C205265
     0634 636F7264
     0638 20547970
     063C 650D00
986  063F 52656164  Msg9:   dcb     'Read Error ... Non ASCII character', Cr, 00
     0643 20457272
     0647 6F72202E
     064B 2E2E204E
     064F 6F6E2041
     0653 53434949
     0657 20636861
     065B 72616374
     065F 65720D00
987  0663 57726974  Msg10:  dcb     'Write Error ... Incorrect Address Length', Cr, 00
     0667 65204572
     066B 726F7220
     066F 2E2E2E20
```

TEST.ASM

```
#################################################################
#                                                               #
#     Archimedes 8096/196 Assembler V1.95/MD2        13/Jul/92  04:04:38   #
#                                                               #
#      Source   =   test.asm                                    #
#      List     =   test.lst                                    #
#      Object   =   test.obj                                    #
#      Options  =   s                                           #
#                                                               #
#                                                               #
#                    (c) Copyright Archimedes Software Inc. 1988 #
#################################################################
```

```
                ;*****************************************************************
                ;*                                                               *
                ;*                 S O L A R   F L U X   M O N I T O R           *
                ;*                                                               *
                ;*            A D V A N C E D   L A N G M U I R   P R O B E      *
                ;*                                                               *
                ;*                          S F M / A L P                        *
                ;*                                                               *
                ;*                                                               *
                ;*                ROM, RAM and Telemetry Test Routines           *
                ;*                                                               *
                ;*****************************************************************

 0000                   Name    Test
 0000                   Rseq    Test

                ;                       Assembler Directives
 0000                   LSTCOD  +
 0000                   LSTWID  -
 0000                   LSTPAG  +
 0000                   LSTXRF
 0000                   PAGSIZ  52
```

30
31
32

```
                ;           ************************************************
                ;           *                                              *
                ;           *                                              *
                ;           *                  P U B L I C S               *
                ;           *                                              *
                ;           *                                              *
                ;           ************************************************

0000            Public      ROM_Test, ZRAM_Test, Telem_Test
```

```
46
47                ;          ************************************************
48                ;          *                                              *
49                ;          *                                              *
50                ;          *              E X T E R N A L S               *
51                ;          *                                              *
52                ;          *                                              *
53                ;          ************************************************
54
55  0000          Extern     Send_Msg, Delay, Send_Hex_Word, Xmt_Char
56  0000          Extern     EEPROM_start, EEPROM_stop, ZRAM_start, ZRAM_stop
57  0000          Extern     Rom_CS, Loader_CS, Telem
58
59
```

```
                ;
                ;               ***************************************************
                ;               *                                                 *
                ;               *      R E G I S T E R   D E F I N I T I O N S    *
                ;               *                                                 *
                ;               ***************************************************


                $REG8096.INC                    ;INCLUDE FILE FOR 8096 REGISTERS
                ;       - REG8096.INC -
                ;
                ;       Include file containing 8096 register definitions
                ;

                ;
                ;       8096 C Compiler Working registers
                ;
  001A          IR_BASE DEFINE  01AH            ; Register base address

  001A          R8      DEFINE  (IR_BASE)
  001B          R9      DEFINE  (IR_BASE+1)

  001C          R0      DEFINE  (IR_BASE+2)
  001D          R1      DEFINE  (IR_BASE+3)
  001E          R2      DEFINE  (IR_BASE+4)
  001F          R3      DEFINE  (IR_BASE+5)
  0020          R4      DEFINE  (IR_BASE+6)
  0021          R5      DEFINE  (IR_BASE+7)
  0022          R6      DEFINE  (IR_BASE+8)
  0023          R7      DEFINE  (IR_BASE+9)

                ;
                ;       Special Function Registers
                ;
  0018          SP              DEFINE  18H
  0018          STACKPOINTER    DEFINE  18H

  0002          AD_RESULT_LO    DEFINE  02H
  0003          AD_RESULT_HI    DEFINE  03H
  0002          AD_COMMAND      DEFINE  02H

  0003          HSI_MODE        DEFINE  03H
  0004          HSI_TIME        DEFINE  04H
  0006          HSI_STATUS      DEFINE  06H

  0004          HSO_TIME        DEFINE  04H
  0006          HSO_COMMAND     DEFINE  06H
```

```
0007        SBUF          DEFINE  07H
0007        SBUF_RX       DEFINE  07H
0007        SBUF_TX       DEFINE  07H

0008        INT_MASK      DEFINE  08H
0009        INT_PEND      DEFINE  09H

000A        WATCHDOG      DEFINE  0AH

000A        TIMER1        DEFINE  0AH
000C        TIMER2        DEFINE  0CH

000E        BAUD_RATE     DEFINE  0EH

000E        IO_PORT0      DEFINE  0EH
000F        IO_PORT1      DEFINE  0FH
0010        IO_PORT2      DEFINE  10H

0011        SP_STAT       DEFINE  11H
0011        SP_CON        DEFINE  11H

0015        IOS0          DEFINE  15H
0016        IOS1          DEFINE  16H

0015        IOC0          DEFINE  15H
0016        IOC1          DEFINE  16H

0017        PWM_CONTROL   DEFINE  17H

            $DEF8096.INC
            ;       - DEF8096.INC -
            ;
            ;       Include file for getting a nonbanked model
            ;

0000        banking       DEFINE  0



            $SFMALP.INC
                                          ; 80C196 Special Function Registers
0012        INT_PEND1     DEFINE          12h
0013        INT_MASK1     DEFINE          13h
0014        WSR           DEFINE          14h
0017        IOS2          DEFINE          17h
0017        IOC2          DEFINE          17h

                                          ; Registers to be used in assembly routines
001A        Ax            DEFINE          1Ah
```

```
001B        Ah          DEFINE      1Bh
001A        Al          DEFINE      1Ah
001C        Bx          DEFINE      1Ch
001D        _Bh         DEFINE      1Dh
001C        Bl          DEFINE      1Ch
001E        Cx          DEFINE      1Eh
001F        Ch          DEFINE      1Fh
001E        Cl          DEFINE      1Eh
0020        Dx          DEFINE      20h
0021        Dh          DEFINE      21h
0020        Dl          DEFINE      20h
0022        Ex          DEFINE      22h
0023        Eh          DEFINE      23h
0022        El          DEFINE      22h
0024        Fx          DEFINE      24h
0025        Fh          DEFINE      25h
0024        Fl          DEFINE      24h

0000        ZERO        DEFINE      00h ; To be used in comparisons

                                    ; Serial Port, Jiffly Clock, and Flags
0026        Rcv_In_Ptr  DEFINE      26h
0028        Rcv_Out_Ptr DEFINE      28h
002A        Xmt_In_Ptr  DEFINE      2Ah
002C        Xmt_Out_Ptr DEFINE      2Ch

002E        Jiffy_Clock DEFINE      2Eh

0030        Serial_Flags DEFINE     30h
0031        Flags       DEFINE      31h
0032        Flags2      DEFINE      32h
```

```
                ;               ***************************************************
                ;               *                                                 *
                ;               *                                                 *
                ;               *                    E Q U A T E S                *
                ;               *                                                 *
                ;               *                                                 *
                ;               ***************************************************


                                        ;   Serial Flag Register bit locations
0000            Xmt_Active:     equ     0       ; Bit 0 = Xmt Active
0001            Xmt_Full:       equ     1       ; Bit 1 = Xmt Full
0002            Xmt_Empty:      equ     2       ; Bit 2 = Xmt Empty
0003            Rcv_Full:       equ     3       ; Bit 3 = Rcv Full
0004            Rcv_Empty:      equ     4       ; Bit 4 = Rcv Empty
0005            CR_Detected:    equ     5       ; Bit 5 = CR Detected
                                                ; Bit 6 =
                                                ; Bit 7 =


                                        ;   Flags register assignment
0000            Timer1_Int:     equ     0       ; Bit 0 = Timer 1 Interrupt Occured
0001            Jiffy_Override: equ     1       ; Bit 1 = Software Enable of Jiffy Timer
                                                ; Bit 2 =
                                                ; Bit 3 =
                                                ; Bit 4 =
                                                ; Bit 5 =
                                                ; Bit 6 =
                                                ; Bit 7 =


                                        ; Flags 2 Register assignment
                                                ; Bit 0 =
                                                ; Bit 1 =
                                                ; Bit 2 =
                                                ; Bit 3 =
                                                ; Bit 4 =
                                                ; Bit 5 =
                                                ; Bit 6 =
                                                ; Bit 7 =


                                        ;  IO_PORT2 pin assignments (input port)
                                                ; Bit 0 =
                                                ; Bit 1 =
                                                ; Bit 2 =
0003            Accel:          equ     3       ; Bit 3 =
```

```
0004        T_Full:     equ     4       ; Bit 4 =
                                        ; Bit 5 =
                                        ; Bit 6 =
                                        ; Bit 7 =


                                ; ASCII characters
000D        Cr:         equ     13      ; Carriage Return
000A        Lf:         equ     10      ; Line Feed
0008        Bs:         equ     08      ; Back Space
007F        Del:        equ     7Fh     ; Delete
0007        Bel:        equ     07      ; Bell
000C        FF:         equ     12      ; Form Feed
0009        Ht:         equ     9       ; Horizontal Tab
000B        Vt:         equ     11      ; Vertical Tab
0020        Sp:         equ     32      ; Space

0000        Port1_init:         equ     000h    ; No Instrument Selected
00FF        Port2_init:         equ     0FFh    ; All Inputs HIGH
```

262

```
                ;*****************************************************************************
                ;*                                                                           *
                ;*                                                                           *
                ;*                          R O M     T E S T                                *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;*  inputs      |    outputs    |    routines called    |   registers altered  *
                ;*****************************************************************************
                ;*   ZRAM_CS    |   ZRAM_CS     |      Send_Msg         |          Ax          *
                ;*              |               |                       |          Bx          *
                ;*              |               |                       |          Cx          *
                ;*****************************************************************************

                ;               This routine verifys the Rom to the checksum value stored
                ;       in the zero power ram.  It is normal for this test to fail the
                ;       first time a new Rom is run.

0000            ROM_Test:       equ     $

0000 FA                 di                              ; disable interrupts
0001 A102001E           ld      Cx, #EEPROM_start + 2   ; rom starting address
0005 011A               clr     Ax                      ; clear result register


0007 661F1A     RT_10:  add     Ax, [Cx]+
000A 8900001E           cmp     Cx, #EEPROM_stop        ; check for the end of the rom
000E D7F7               jne     RT_10

0010 FB                 ei                              ; re-enable interrupts
0011 8B010000           cmp     Ax, Rom_CS              ; compair calc cksum to stored cksum,
     0015 1A
0016 DF0E               je      RT_20                   ;     jump if ok.
0018 C3010000           st      Ax, Rom_CS              ; If Error, Store new cksum
     001C 1A
001D A106001A           ld      Ax, #6                  ; msg #6  'ROM cksum fail'
0021 EFFFFE             call    Send_Msg
0024 2007               sjmp    RT_30

0026 A109001A   RT_20:  ld      Ax, #9                  ; 'ROM ok'
002A EFFFFE             call    Send_Msg

002D A3010000   RT_30:  ld      Ax, Rom_CS              ; verify ROM_CS equals loader CS
     0031 1A
0032 8B010000           cmp     Ax, Loader_CS
     0036 1A
```

```
0037 DF07              je      RT_40
0039 A115001A          ld      Ax, #21
003D EFFFFE            call    Send_Msg

0040 F0        RT_40:  ret
```

```
316
317                 ;*****************************************************************************
318                 ;*                                                                           *
319                 ;*                                                                           *
320                 ;*                          Z R A M     T E S T                              *
321                 ;*                                                                           *
322                 ;*                                                                           *
323                 ;*****************************************************************************
324                 ;* inputs      |    outputs     |    routines called    |   registers altered *
325                 ;*****************************************************************************
326                 ;*   ZRAM_CS   |   ZRAM_CS      |       Send_Msg        |          Ax         *
327                 ;*             |                |                       |          Bx         *
328                 ;*             |                |                       |          Cx         *
329                 ;*****************************************************************************
330
331                 ;               This routine checks the zero power ram byte by byte with an
332                 ;       alternating bit pattern to verify that eack ram location is ok.  It
333                 ;       also calculates the 16 bit checksum of the zero power ram and verifys
334                 ;       that agents what is stored.  The test performed is non-destructive.
335
336  A55A           ZRAM_Check_Word := 0A55Ah
337
338  0041           ZRAM_Test:      equ     $
339  0041 FA                 di                              ; disable interrupts
340  0042 A100001E           ld      Cx, #ZRAM_start         ; start of Zram
341
342  0046 A21E1A    ZT_10:   ld      Ax, [Cx]                ; read a character
343  0049 A15AA51C           ld      Bx, #ZRAM_Check_Word    ; Bx = Alternating bit pattern
344  004D C21E1C             st      Bx, [Cx]                ; write alternating bit pattern
345  0050 8A1E1C             cmp     Bx, [Cx]                ; verify pattern
346  0053 C21F1A             st      Ax, [Cx]+               ; restore origonal data
347  0056 D70F               jne     ZT_30                   ; ck compair for ram error
348  0058 8900001E           cmp     Cx, #ZRAM_stop          ; test for end of Zram
349  005C DEE8               jlt     ZT_10
350
351  005E FB                 ei                              ; re-enable interrupts
352  005F A108001A           ld      Ax, #8                  ; 'ZRAM pass' msg #8
353  0063 EFFFFE             call    Send_Msg
354  0066 F0                 ret                             ; exit routine. Zram passed ok
355
356
357  0067 FB        ZT_30:   ei
358  0068 6902001E           sub     Cx, #2
359  006C A01E1A             ld      Ax, Cx
360  006F EFFFFE             call    Send_Hex_Word
361  0072 B1201A             ldb     Al, #' '
362  0075 EFFFFE             call    Xmt_Char
363  0078 A105001A           ld      Ax, #5                  ; msg #5    'Zram Failed'
364  007C EFFFFE             call    Send_Msg
```

```
365  007F F0          ret
366
```

```
368
369                 ;*****************************************************************************
370                 ;*                                                                           *
371                 ;*                                                                           *
372                 ;*                          T E L E M    T E S T                          *
373                 ;*                                                                           *
374                 ;*                                                                           *
375                 ;*****************************************************************************
376                 ;* inputs      |    outputs    |    routines called    |   registers altered *
377                 ;*****************************************************************************
378                 ;*   T_Full     |    Telem      |      Send_Msg         |          Ax         *
379                 ;*              |               |                       |                     *
380                 ;*              |               |                       |                     *
381                 ;*****************************************************************************
382
383                 ;               This routine fills the telemetry buffer with A5 hex until it
384                 ;       is full and then checks that it begins to empty within 25 mSec.  If
385                 ;       it does then an 'OK' message is sent, if not then a failure message.
386                 ;       If the telemetry buffer is full upon entry into this routine then a
387                 ;       message to that extent is sent.  Upon filling a 512 byte buffer it
388                 ;       will require 1.625 sec for telemetry to empty it
389
390  01A5           Telem_Test_Word  :=  01A5h
391
392  0080           Telem_Test:
393
394  0080 3C1008            jbs     IO_PORT2, T_Full, TT_10        ; jump if telemetry buff not full
395  0083 A10F001A          ld      Ax, # 15                       ; send telemetry full message
396  0087 EFFFFE            call    Send_Msg
397  008A F0                ret
398
399  008B C81C      TT_10:  push    Bx
400  008D A188131C          ld      Bx, #5000                      ; full telem buff w/ up to 5000
401  0091 A1A5011A          ld      Ax, #Telem_Test_Word           ;  words, (512 should be enough)
402
403  0095 34100F    TT_20:  jbc     IO_PORT2, T_Full, TT_30        ; exit when telem is full
404  0098 C3010000          st      Ax, Telem
     009C 1A
405  009D 051C              dec     Bx                             ; continue looping until buff is
406  009F D7F4              jne     TT_20                          ;   full.
407                                                                ; if unable to fill Telem Buff
408  00A1 A112001A          ld      Ax, #18                        ;   send error message.
409  00A5 2010              sjmp    TT_40
410
411  00A7 A119001A  TT_30:  ld      Ax, #25                        ; wait for 25 mSec
412  00AB EFFFFE            call    Delay
413
414  00AE A110001A          ld      Ax, #16                        ; if buffer not empty send pass
415  00B2 3C1002            jbs     IO_PORT2, T_Full, TT_40        ;    message, if still full send
```

```
00B5 071A                inc     Ax                ;   send failure msg

00B7 EFFFFE     TT_40:   call    Send_Msg
00BA CC1C                pop     Bx
00BC F0                  ret

00BD                     END
```

```
Errors:  None          ########
Bytes:   189           # Test #
CRC:     8CB5          ########
```

Symbol and Cross Reference Table

Symbol	Value	Type	Defline	Refline						
Segment Definitions										
Test	00BD	S00R	15							
External Symbols										
Delay	0000	X01	55	412						
EEPROM_start	0000	X04	56	288						
EEPROM_stop	0000	X05	56	293						
Loader_CS	0000	X09	57	308						
Rom_CS	0000	X08	57	297	299	307				
Send_Hex_Word	0000	X02	55	360						
Send_Msg	0000	X00	55	301	305	311	353	364	396	418
Telem	0000	X0A	57	404						
Xmt_Char	0000	X03	55	362						
ZRAM_start	0000	X06	56	340						
ZRAM_stop	0000	X07	56	348						
Public Symbols										
ROM_Test	0000	R00E	285	43						
Telem_Test										

	0080	R00E	392	43
ZRAM_Test	0041	R00E	338	43

Local Symbols
==========

AD_COMMAND	0002	A#	101								
AD_RESULT_HI	0003	A#	100								
AD_RESULT_LO	0002	A#	99								
Accel	0003	A	240								
Ah	001B	A#	159								
Al	001A	A#	160	361							
Ax	001A	A#	158	289	292	297	299	300	304	307	
				308	310	342	346	352	359	363	
				395	401	404	408	411	414	416	
BAUD_RATE	000E	A#	122								
Bel	0007	A	252								
Bl	001C	A#	163								
Bs	0008	A	250								
Bx	001C	A#	161	343	344	345	399	400	405	419	
CR_Detected	0005	A	209								
Ch	001F	A#	165								
Cl	001E	A#	166								
Cr	000D	A	248								
Cx	001E	A#	164	288	292	293	340	342	344	345	

				346	348	358	359
Del	007F	A	251				
Dh	0021	A#	168				
Dl	0020	A#	169				
Dx	0020	A#	167				
Eh	0023	A#	171				
El	0022	A#	172				
Ex	0022	A#	170				
FF	000C	A	253				
Fh	0025	A#	174				
Fl	0024	A#	175				
Flags	0031	A#	188				
Flags2	0032	A#	189				
Fx	0024	A#	173				
HSI_MODE	0003	A#	103				
HSI_STATUS	0006	A#	105				
HSI_TIME	0004	A#	104				
HSO_COMMAND	0006	A#	108				
HSO_TIME	0004	A#	107				
Ht	0009	A	254				
INT_MASK	0008	A#	114				
INT_MASK1	0013	A#	152				
INT_PEND	0009	A#	115				

INT_PEND1	0012	A#	151								
IOC0	0015	A#	134								
IOC1	0016	A#	135								
IOC2	0017	A#	155								
IOS0	0015	A#	131								
IOS1	0016	A#	132								
IOS2	0017	A#	154								
IO_PORT0	000E	A#	124								
IO_PORT1	000F	A#	125								
IO_PORT2	0010	A#	126	394	403	415					
IR_BASE	001A	A#	79	81	82	84	85	86	87	88	
				89	90	91					
Jiffy_Clock	002E	A#	185								
Jiffy_Override	0001	A	216								
Lf	000A	A	249								
PWM_CONTROL	0017	A#	137								
Port1_init	0000	A	258								
Port2_init	00FF	A	259								
R0	001C	A#	84								
R1	001D	A#	85								
R2	001E	A#	86								
R3	001F	A#	87								

R4	0020	A‡	88	
R5	0021	A‡	89	
R6	0022	A‡	90	
R7	0023	A‡	91	
R8	001A	A‡	81	
R9	001B	A‡	82	
RT_10	0007	R00	292	294
RT_20	0026	R00	304	298
RT_30	002D	R00	307	302
RT_40	0040	R00	313	309
Rcv_Empty	0004	A	208	
Rcv_Full	0003	A	207	
Rcv_In_Ptr	0026	A‡	180	
Rcv_Out_Ptr	0028	A‡	181	
SBUF	0007	A‡	110	
SBUF_RX	0007	A‡	111	
SBUF_TX	0007	A‡	112	
SP	0018	A‡	96	
SP_CON	0011	A‡	129	
SP_STAT	0011	A‡	128	
STACKPOINTER	0018	A‡	97	
Serial_Flags	0030	A‡	187	

Sp	0020	A	256			
TIMER1	000A	A#	119			
TIMER2	000C	A#	120			
TT_10	008B	R00	399	394		
TT_20	0095	R00	403	406		
TT_30	00A7	R00	411	403		
TT_40	00B7	R00	418	409	415	
T_Full	0004	A	241	394	403	415
Telem_Test_Word	01A5	A	390	401		
Timer1_Int	0000	A	215			
Vt	000B	A	255			
WATCHDOG	000A	A#	117			
WSR	0014	A#	153			
Xmt_Active	0000	A	204			
Xmt_Empty	0002	A	206			
Xmt_Full	0001	A	205			
Xmt_In_Ptr	002A	A#	182			
Xmt_Out_Ptr	002C	A#	183			
ZERO	0000	A#	177			
ZRAM_Check_Word	A55A	A	336	343		
ZT_10	0046	R00	342	349		

ZT_30	0067	R00	357	347
_Bh	001D	A#	162	
banking	0000	A#	145	

Macro Definitions
==========

CROSS-REF.

```
Archimedes Universal Linker V4.11/DOS          14/Jul/92  16:28:17

   Target CPU    =  8096
   List file     =  sfmalp.out
   Output file 1 =  sfmalp.hex
   Output format =  intel-standard
   Command line  =  -f sfmalp.xcl (-c8096 sfm-alp com_proc modes
                    modes2 tm modes_tm crc io calib sfmalp.obj
                    serial.obj isr.obj timers.obj test.obj printf
                    putchar.obj getchar.obj cl8096
                    -Z(CODE)LOAD_ISR,ROM,Ser,ISR,Timers,Test,CODE,
                    RCODE,CDATA,ZVECT,CSTR,CCSTR,CSTART,CONST=4000

                    -Z(DATA)CSTACK,ZRAM,DATA,IDATA,UDATA,ECSTR,
                    WCSTR,TEMP,Telem_Buff=C000
                    -Fintel-standard -o sfmalp.hex -xsmi -l
                    sfmalp.out -p58)
```

(c) Copyright Archimedes Software Inc. 1988

```
*****************************************
*                                       *
*            CROSS REFERENCE            *
*                                       *
*****************************************
```

Program entry at : B452 Relocatable, from module : CSTARTUP

```
*****************************************
*                                       *
*              MODULE MAP               *
*                                       *
*****************************************
```

FILE NAME : sfm-alp.rl5
PROGRAM MODULE, NAME : sfm-alp

SEGMENTS IN THE MODULE
==========

CODE
Relative segment, address : 4738 - 4964

ENTRIES	ADDRESS	REF BY MODULE
send_hello	4931	Not referred to
main	4738	sfmalp
		CSTARTUP
init_system	48A9	Not referred to

CSTR
Relative segment, address : A720 - A826

UDATA
Relative segment, address : D154 - D157

ENTRIES	ADDRESS	REF BY MODULE
D_count	D154	com_proc
		modes
		modes2
		tm
		modes_tm
		crc
		io
		calib
DEBUG_LEVEL	D156	com_proc
		modes
		modes2
		tm
		modes_tm
		crc
		io
		calib

****

FILE NAME : com_proc.rl5
PROGRAM MODULE, NAME : com_proc

SEGMENTS IN THE MODULE
==========

CODE
Relative segment, address : 4966 - 57CF

ENTRIES	ADDRESS	REF BY MODULE
uppercase	553B	Not referred to
proc_instr_params	53AF	calib
hex_conv	5597	calib
dec_conv	5671	calib
com_proc	4966	sfm-alp
c_tstop	4C55	Not referred to
c_tstart	4C39	Not referred to
c_tm_test	5011	Not referred to
c_status	4C80	sfm-alp
c_smode	4C71	Not referred to
c_set_vern	4EDA	Not referred to
c_set_va	4E63	Not referred to
c_set_pg	4F25	Not referred to
c_set_grid	4EA0	Not referred to
c_ser_tm	4DFD	Not referred to
c_sample	4F7C	Not referred to
c_reset	4C10	Not referred to
c_relay	5023	Not referred to
c_loader	539D	Not referred to
c_help	4B6A	Not referred to
c_debug	52FC	Not referred to
c_convtest2	51AA	Not referred to
c_convtest	50BE	Not referred to
c_clrscr	52E5	sfm-alp
bin_conv	5722	calib

CSTR
Relative segment, address : A828 - AFA0

UDATA
Relative segment, address : D158 - D1C3

ENTRIES	ADDRESS	REF BY MODULE
word_out	D1C2	Not referred to
real_out	D1BE	Not referred to
comm_v	D158	Not referred to
comm_c	D1BC	Not referred to

CONST
Relative segment, address : B460 - B527

ENTRIES	ADDRESS	REF BY MODULE
commands	B460	Not referred to

****

FILE NAME : modes.rl5
PROGRAM MODULE, NAME : modes

SEGMENTS IN THE MODULE

```
   ======================
  )DE
   Relative segment, address : 57D0 - 5DA2
          ENTRIES                ADDRESS          REF BY MODULE
          vconverge               5D21            com_proc
                                                  modes2
          set_mode_pvg            5B97            com_proc
                                                  modes2
          null_mode               5A38            Not referred to
          next_mode_step          5885            sfm-alp
          init_modes              57D0            sfm-alp
                                                  com_proc
          hskp_mode               5A99            Not referred to
    ---------------------------------------------------------------------
CSTR
  Relative segment, address : AFA2 - B2F1
    ---------------------------------------------------------------------
UDATA
  Relative segment, address : D1C4 - D1CD
          ENTRIES                ADDRESS          REF BY MODULE
          vern_sar_step           D1C8            sfm-alp
                                                  com_proc
                                                  modes2
                                                  tm
                                                  modes_tm
                                                  io
          vern_conv_val           D1C6            sfm-alp
                                                  com_proc
                                                  modes2
                                                  tm
                                                  modes_tm
                                                  io
          va_sar_step             D1CC            sfm-alp
                                                  com_proc
                                                  modes2
                                                  tm
                                                  modes_tm
                                                  io
          va_conv_val             D1CA            sfm-alp
                                                  com_proc
                                                  modes2
                                                  tm
                                                  modes_tm
                                                  io
          pg_converged            D1C4            sfm-alp
                                                  com_proc
                                                  modes2
                                                  tm
                                                  modes_tm
                                                  io
    ---------------------------------------------------------------------
CONST
  Relative segment, address : B528 - BB47
          ENTRIES                ADDRESS          REF BY MODULE
          modelist                B528            sfm-alp
                                                  com_proc
```