NASA CR-172119 d.

Part 1 + Part 2

STAND ALONE PRESSURE MEASUREMENT DEVICE (SAPMD) FOR THE SPACE SHUTTLE ORBITER

by

Bill Tomlinson

FINAL REPORT
NASA Contract NAS9-17601
SwRI Project 15-1062

for

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas 77058

January 1989

INTRODUCTION

This document presents the final technical report for the development and delivery of a Stand Alone Pressure Measurement Device (SAPMD) and associated ground support equipment. This program was developed for the NASA/Johnson Space Center under NASA contract NAS9-17601 (SwRI Project 15-1062).

The data and documentation contained herein are the results of the development and successful completion of this contract.

Background

This program fulfilled the need to measure pressure at the surface of the thermal protective system tile on the space shuttle Orbiter during ascent, and in order to avoid the extensive impact associated with wiring the measurement into the Orbiter data system, the measurement device must be completely stand-alone and incorporate its own power supply and data recording facility. The device must be small enough to be mounted under the thermal protection system tiles and must be rugged enough to withstand the environments it will encounter at the bond line of the tiles throughout an Orbiter mission. It must be failsafe and data recorded during ascent must be recoverable after the mission without removal of the device.

Specifications

The SAPMD shall measure ambient pressure at the surface of the Orbiter TPS in the range of 0-15 pounds per square inch absolute (PSIA). Measurement will begin at solid rocket booster (SRB) ignition as sensed by appropriate vibration sensing elements in the SAPMD. Pressure and corresponding real-time data are to be recorded every one tenth second for 140 seconds and at the end of the recording period, the operation will be discontinued with the data preserved for interrogation subsequent to Orbiter re-entry and landing.

The type and size of the battery shall be such as to allow the vibration sensing elements and a real-time clock to be initialized a minimum of 30 days prior to launch and still provide power as necessary to perform the 140 second data recording period after SRB ignition. Battery installation shall be in such a manner as to allow battery replacement without removing the SAPMD from its position or removing more than one TPS tile.

The SAPMD must be mounted in specific locations under tiles of the Orbiter TPS. To accommodate such mounting, the absolute maximum physical dimensions must not exceed 6.0 inches in length, 1.5 inches in width and 0.4 inches in height, and the device shall be of such configuration that it can be bonded to the Orbiter skin at the joint line of two TPS tiles with the pressure sensing port at the surface of the tile. The SAPMD must remain operational in the temperature range of -40 to +85°C and survive storage temperatures of -55 to +125°C. The pressure port must withstand 934°C without causing damage to the TPS during entry and must remain functional at 262°C during ascent.

The accuracy of the pressure measurement must be plus or minus one-half pound per square inch absolute over a temperature range of 0 to +36°C.

Conclusion

All of the above specifications have been met and verified by prototype testing and is documented in the enclosed test data.

Four flight-qualified models were fabricated and of these, two have been delivered and successfully flown in the cargo bay of STS-26.

A contract modification changed the delivery of four flight models to two while modifying the remaining two for use in the nozzle bearing area of the SRB during a ground test at the Morton Thiokol site in Utah.

INDEX

CEI SPECIFICATION

Specification No. 1062-CEI-01
Release Date: 19 October 1987

CONTRACT END ITEM SPECIFICATION

PERFORMANCE/DESIGN

AND

PROJECT CONFIGURATION

REQUIREMENTS

SE-176TA

STAND-ALONE PRESSURE MEASUREMENT DEVICE FOR THE SPACE SHUTTLE ORBITER

CONTRACT NUMBER NAS 9-17601

Approved by: Rex R. Ritz
JSC Contracting
Officer
Houston, Texas

W. C. Gibson

Approved by: William C. Gibson
Southwest Research
Institute
6220 Culebra Road
San Antonio, Texas

Approval date: ______________

Approval date: 20 Oct 87

REVISION RECORD

REV	SCN. NUMBER	PAGES AFFECTED	PARAGRAPHS AFFECTED	DATE	APPROVAL

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Scope

This specification establishes the requirements for complete identification and acceptance of a Stand-Alone Pressure Measurement Device (SAPMD) for the Space Shuttle Orbiter to be formally accepted by the Manned Spacecraft Center (MSC).

1.2 Engineering Baseline

The engineering baseline shall be established by a Critical Design Review (CDR) for this Contract End Item (CEI). All units of this CEI, regardless of intended use, shall be manufactured and accepted to the configuration defined by this pseciflcation and formally approved Engineering Change Proposals (ECP's)/Specification Change Notices (SCN's).

2.0 APPLICABLE DOCUMENTS

The following documents of the exact issue shown form a part of this specification to the extent specified herein. In the event of conflict between this specification and documents referenced herein, this specification shall take precedence.

Specifications-JSC

NHB 5300.4 (3A-1)	Requirements for Soldering of Electrical Connections
NHB 5300.4 (1D2)	Safety, Reliability, Maintainability, and Quality Provisions for the Space Shuttle Program
NHB 8060.1B	Flammability, Odor, and Offgassing Requirements
NHB 5300.4 (IC)	Inspection System Provisions
JSC 07700, Vol. IV	Configuration Management
JSCM 8080	Criteria and Standards
JSC 02681	Nonmetallic Materials Design Guidelines and Test Data Handbook
JSC-09604B	JSC GFE Materials Selection List and Matrials Documentation Procedures
JSC-SE-R-0006B	NASA/JSC Materials and Processes
JSC 17481	JSC Safety Guidelines Document for Space Shuttle GFE
JSC-SL-E-0002B	Specification, Electromagnetic Interference Characteristics, Requirements for Equipment for the Space Shuttle Program
JSC-SP-T-0023B	Specification, Environmental Acceptance Testing
JSC/MSC-SPEC-M-1A	Marking and Identification
JSC SW-E-0002	Space Shuttle Program GSE General Design Requirements

Specifications-Rockwell

MF-0004-002B	Electrical Design Requirements for Electrical Equipment Utilized on the Space Shuttle Vehicle

Standards-Military

MIL-STD-975E	NASA Standard (EEE) Parts List

3.0 TECHNICAL REQUIREMENTS

3.1 Performance

The Stand Alone Pressure Measure Device (SAPMD) shall measure ambient pressure at the surface of the Orbiter TPS. The measurement range shall be 0 - 15 psia. The measurements shall begin at solid rocket booster (SRB) ignition as sensed by appropriate vibration sensors located within the enclosure incorporating the battery and electronics. Upon sensing SRB ignition, the SAPMD will monitor and record pressure for 140 seconds to a solid state non-volatile memory storage device. At the end of the recording period, the operation will be discontinued with the data preserved for interrogation subsequent to Orbiter entry and landing.

The SAPMD shall have a means to accurately time tag the recorded data in units of 1/2 seconds since January 1. The timekeeping and vibration sensor circuit shall be initialized 30 days before launch. The battery capacity shall be such that this timekeeping can be continued for a minimum of 50 days.

The block diagram shown in Figure 3-1 depicts the method in which the SAPMD shall process and record the pressure nd time data. The heart of the system will be an INTEL 80C31, 8-bit CMOS processor with the program in electrically eraseable programmable prom and the memory device shall be a 64K CMOS electrically eraseable prom capable of 10-year data retention.

The battery supply shall be two each 600 mAH Lithium Thionly Chloride batteries in a removable battery holder.

Data retrieval shall be accomplished with a battery-powered 80C88-based computer. Communication with the SAPMD shall be serial with additional connector pins to provide auxilliary power to the SAPMD.

The SAPMD shall be fabricated to meet the environmental conditions as specified in paragraphs 3.5.1 and 3.5.2 of the contract specification.

3.2 Product Configuration

Figure 3-2 Top Assembly Drawing.

3.2.1 Manufacturing Drawings

The configuration of the SAPMD shall be in accordance with drawing number 15-1062-457, and drawings and engineering data assembled thereunder, including all approved changes thereto. Class II changes to manufacturing drawings are allowable without NASA approval, however they are subject to classification review by NASA.

3.2.2 Government Furnished Property List

NONE

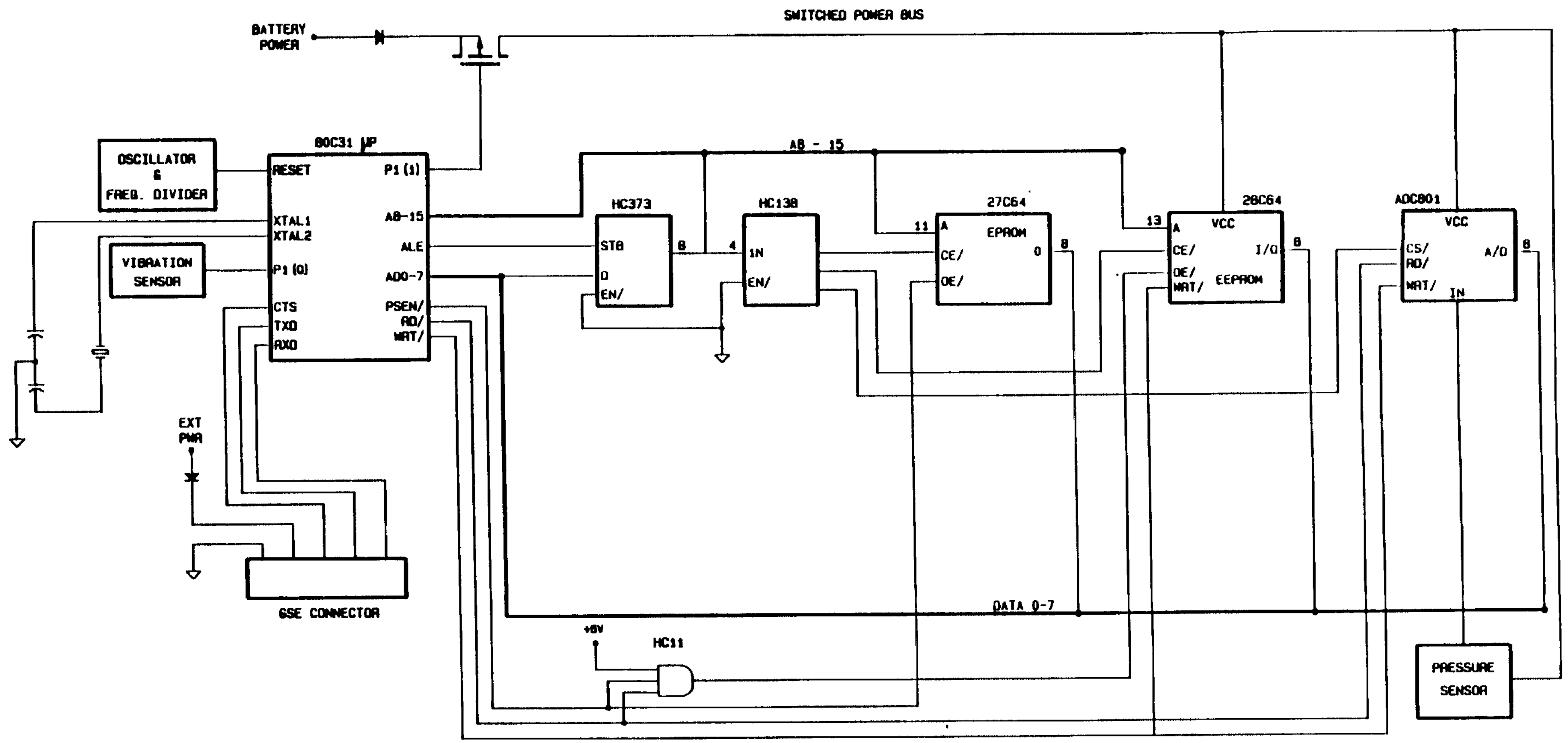


FIGURE 3-1. REVISED SAPMD BLOCK DIAGRAM

SAPMD ASSEMBLY

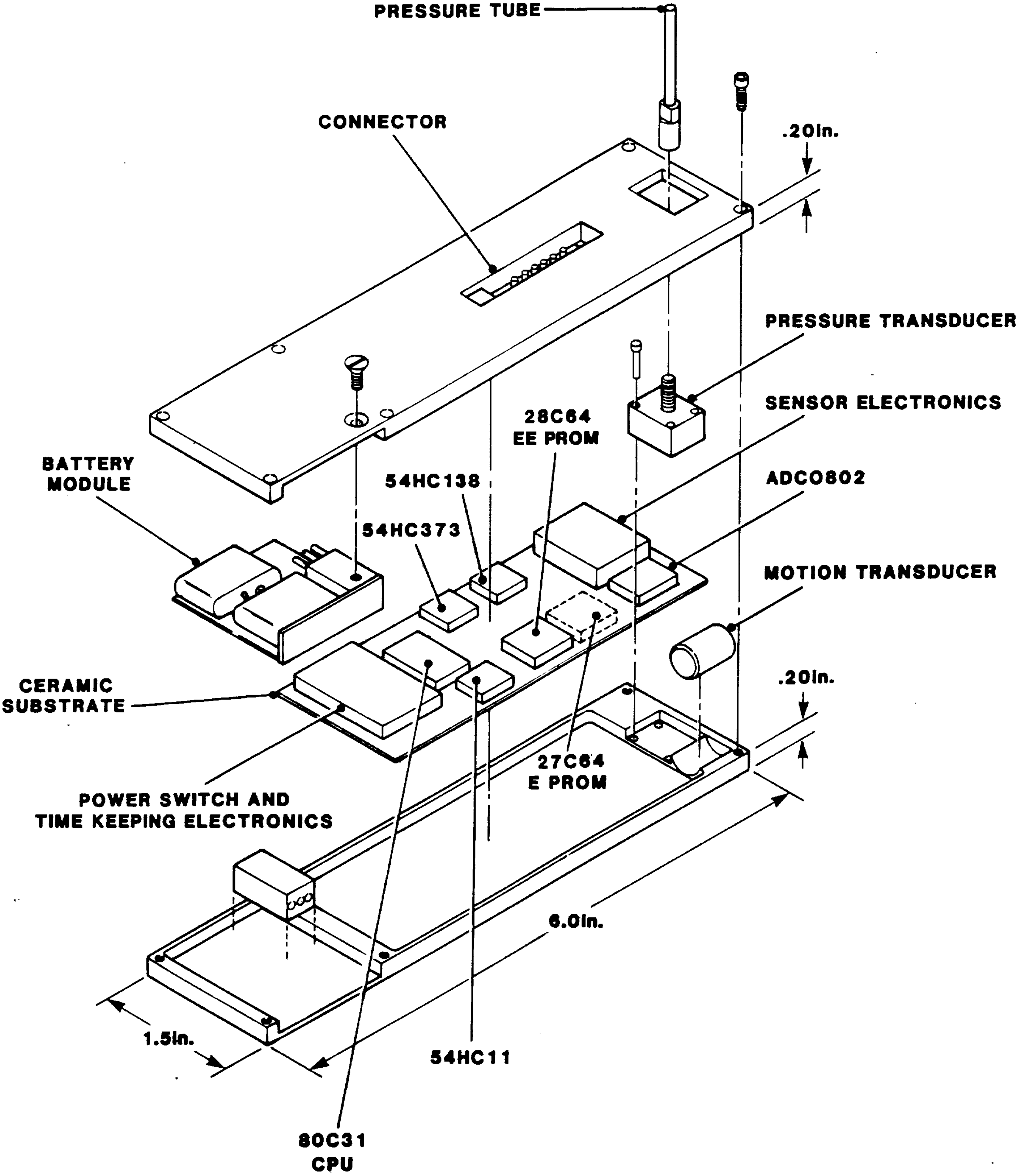


FIGURE 3-2 TOP ASSEMBLY DRAWING

3.2.3 Standards of Manufacturing, Manufacturing Processes, and Production

The applicability of the following publications to the SAPMD may be revised only by engineering changes having prior approval of NASA.

MIL-STD-975F	NASA Standard Electrical, Electronic, and Electromechanical (EEE) Parts List

Specifications-Military

None

Specifications-NASA

JSC/MSFC-SPEC-M-1A	Marking and Identification
JSC-SE-R-0006B	NASA/JSC Requirements for Materials and Processes
JSC-SL-E-0002A	Specification, Electromagnetic Interference Characteristics, Requirements for Equipment for the Space Shuttle Program
JSC-SP-T-0023B	Specification, Environmental Acceptance Testing
JSC SW-E-0002, Rev. B	Space Shuttle Program GSE General Design Requirements

Documents-NASA

JSC 07 700, Vol. IV Rev. B	Space Shuttle Program Configuration Management Requirements (with changes through No. 60)
JSCM 8080	Manned Spacecraft Criteria and Standards
JSC-09604B	JSC GFE Materials Selection List and Materials Documentation Procedures
JSC 17481A	Safety Requirements Document for JSC Space Shuttle Flight Equipment
NHB 5300.4(3A-1)	Requirements for Soldering Electrical Connections
NHB 5300.4(1D2)	Safety, Reliability, Maintainability, and Quality Provisions for the Space Shuttle Program

NHB 8060.1B	Flammability, Odor, and Offgassing Requirements and Test Procedures for Materials, in Environments that Support Combustion
NHB 5300.4(1C)	Inspection System Provisions for Aeronautical and Space System Materials, Parts, Components and Services

Other Standards/Documents

Rockwell MF-0004-002B	Electrical Design Requirements for Electrical Equipment Utilized on the Space Shuttle Vehicle

4.0 QUALITY ASSURANCE

Southwest Research Institute is responsible for accomplishment of each verification required herein.

4.1 Quality Requirements

4.1.1 Applicability of NHB 5300.4 (1D2)

Paragraphs 1D200 and 1D301.6.

4.1.2 Applicability of NHB 5300.4 (3A-1)

A) Chapter 2, all paragraphs.

B) Chapter 3, all paragraphs.

C) Chapter 4, all but paragraphs 3A401, and 3A502.

D) Chapter 8, all paragraphs.

4.1.3 Drawing Compliance

Written verification that the SAPMD has been fabricated, inspected, and tested to the latest applicable drawings identified in 3.2.1 and has incorporated the GFP specified in 3.2.2 will be provided at each Acceptance Review.

4.1.4 Additional Requirements

Paragraph 5.1.3, JSC document 20793.

4.2 Reliability Requirements

A) Design per document JSCM 8080.

B) Design Review (PDR and CDR).

C) Limited life items identification per SwRI document 1062-LL-01.

D) EEE parts per Mil-Std-975F (where possible).

E) Derating per Mil-Std-975F, appendix A.

4.2.1 Additional Requirements

None

4.3 Test Requirements

Per contract NAS9-17601, latest revision.

5.0 PREPARATION FOR DELIVERY

5.1 Containers

Unless otherwise specified, the preservation, packaging, and packing shall be equivalent to the contractor's best commercial practice, provided that this practice will be sufficient to protect the SAPMD against damage during shipment. Exterior containers shall conform to Uniform Freight Classification Rules for rail shipment or National Motor Freight Classification Rules for truck shipment, as applicable.

5.2 Marking

Interior and exterior containers shall be marked in accordance with MIL-STD-129 "Marking for Shipment and Storage".

6.0 NOTES

6.1 Intended Use

The SAPMD, part number 15-1062-900-01, is intended for use in the measurement of ambient air pressure and the recording of that data in the vicinity of the Space Shuttle Orbiter exterior surfaces. Data thus acquired will be transferred to a portable computer system post flight for analysis and archiving.

6.2 Ordering Data

Procurement documents shall specify:

(a) Contract End Item Specification for the Stand-Alone Pressure Measurement Device for the Space Shuttle Orbiter, SwRI Document No. 1062-CEI-01, date 20 October 1987.

(b) Special precautions shall be applied to control of electrostatic discharge during all stages of parts procurement, storage, fabrication and test.

6.3 Definitions

A) SAPMD - Stand Alone Pressure Monitor Device

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

FLIGHT HAZARD EVALUATION OF THE LITHIUM THIONYL CHLORIDE CELL

FLIGHT HAZARD EVALUATION

OF THE

LITHIUM THIONYL CHLORIDE CELL

PURPOSE OF EVALUATIONS

- Temperature Vacuum Test
  - Loss of Hermeticity of Package and Temperature at Which That Loss Occurred
  - Electromechanical Failure
    - Degradation of the Cell's Ability to Supply Power and Temperature at Which That Degradation Occurred
    - Qualitative Rate of Failure Over Time

Temperature Vacuum Test Results

- Temperature Risk of Less Than 5°c/min. Never Caused Violent Rupture of Case

- Cell Continued to Produce Usable Power Even After Encapsulant Failure

Purpose of Evaluations

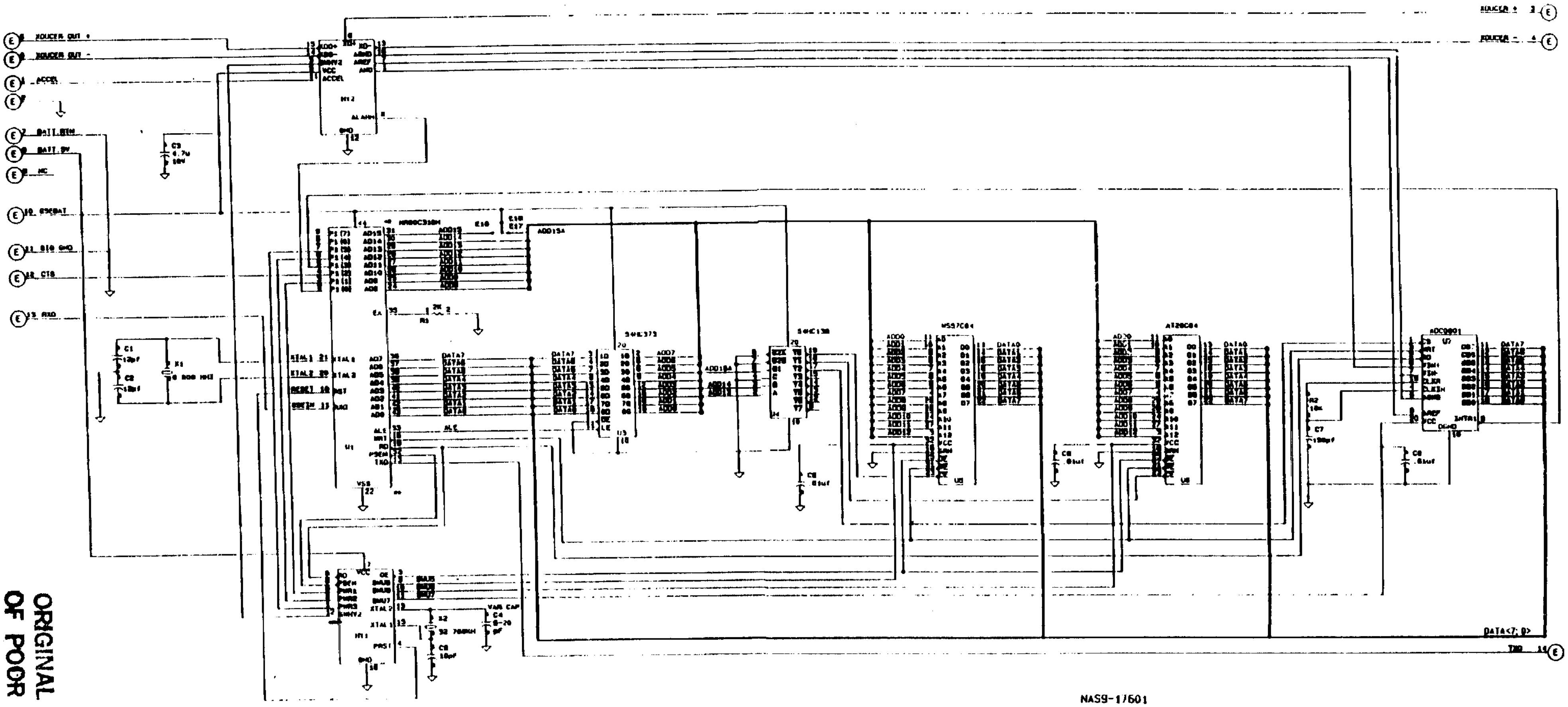
- Short Circuit Test
  - Time Rate of Case Temperature Change
  - Maximum Short Circuit Current
  - The Degradation of the Cell's Ability to Supply Power

Short Circuit Test Results

* Case Temperature Could Exceed 100°c With No Visible Damage To Case and No Loss of Encapsulant Integrity

* Output Current Could Exceed 1.0 Ampere and Cell Could Still Produce Usable Power After Test

SAPMD SCHEMATICS



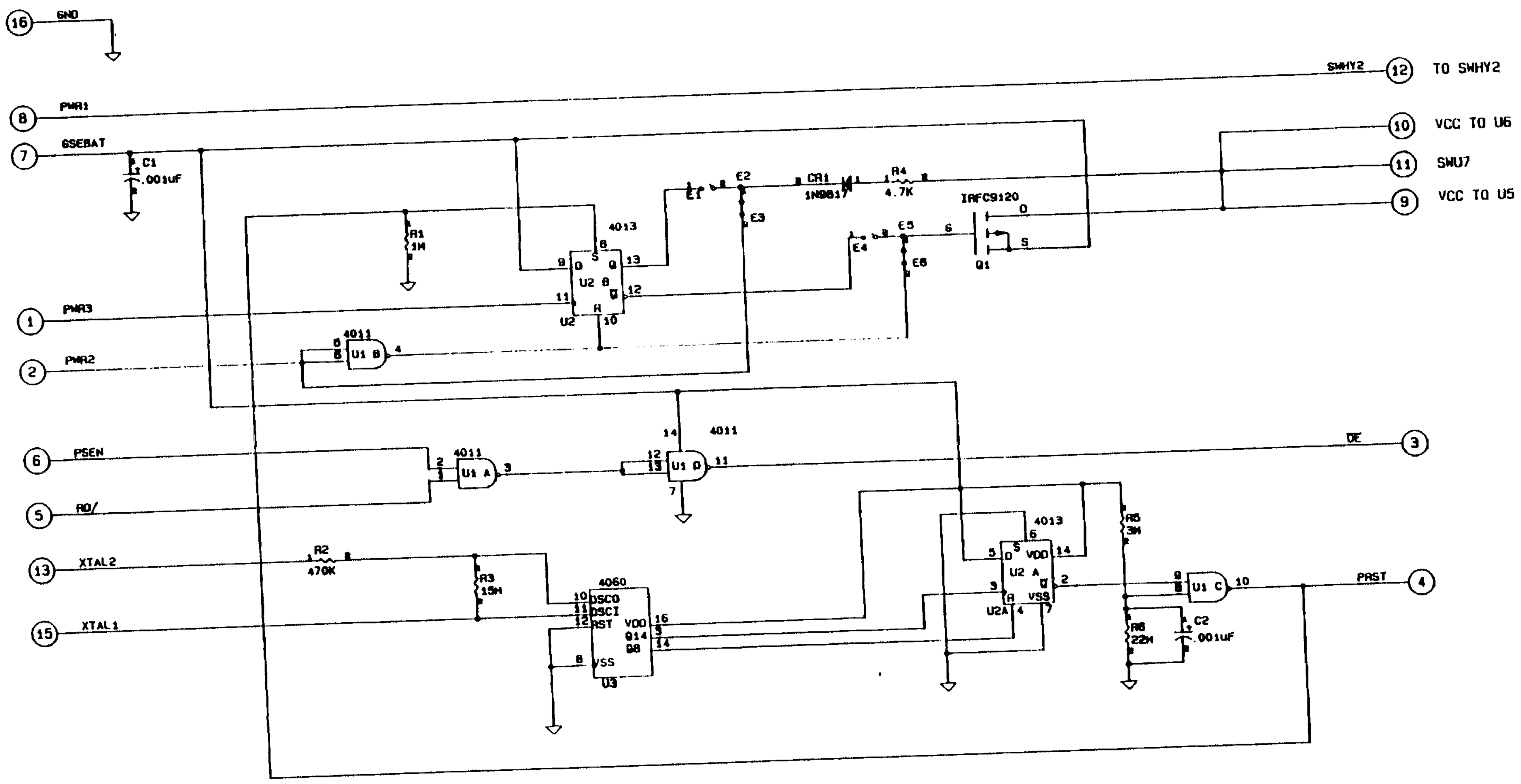
NAS9-17601

SAPMD ELECTRONIC SCHEMATIC

15-1062-301

ORIGINAL PAGE IS OF POOR QUALITY

A CAP. VALUES CHANGED, TITLE CHANGED 4/13/87



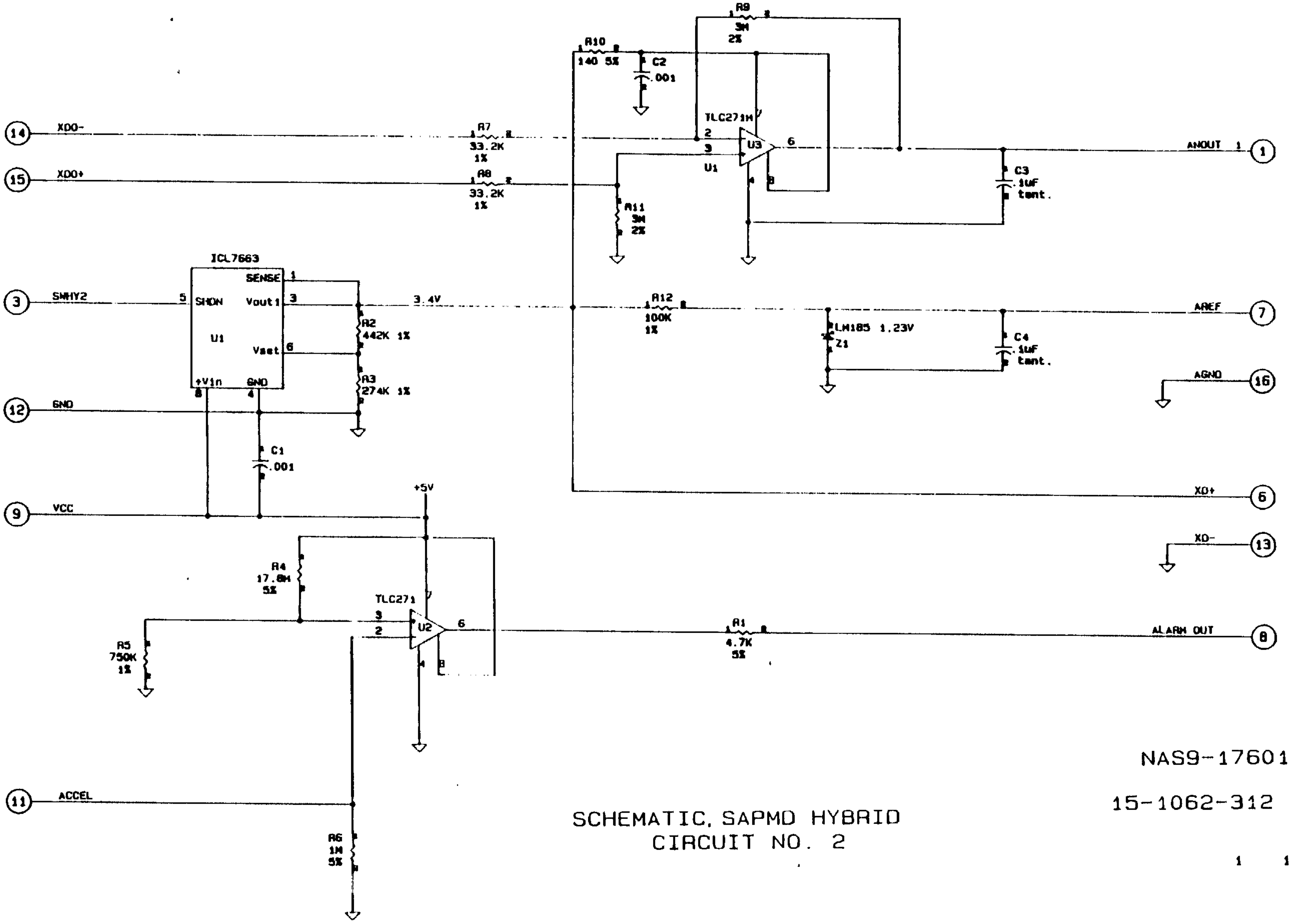
NAS9-17601

15-1062-311 A

SCHEMATIC, SAPMD HYBRID
CIRCUIT NO.' 1

1 1

A R7, R8 VALUE CHG'D., TITLE CHANGED 4/13.



NAS9-17601

15-1062-312 A

SCHEMATIC, SAPMD HYBRID
CIRCUIT NO. 2

1 1

TEST DATA, PROTOTYPE

PROTOTYPE SAPMD TEST RESULTS

- Initial Tests Conducted 3 March 1987 @ NASA Dryden Flight Research Center
  - Results Unacceptable
    - Very High Zero Drift w/Temperature
    - High Pressure Transducer Drift w/Temperature
- Unit Returned to SwRI for Repair/ Calibration
  - Error Sources Analyzed
    - High Zero Drift w/Temperature from Pressure Transducer
    - Large Error Resulting in Temperature Drift of Voltage Regulator
  - Prototype Modified and Recalibrated
    - Pressure Transducer Replaced w/Better Performing Unit
    - Current Limit of Voltage Regulator Raised
    - Thermistor Inserted in Series w/Pressure Transducer
    - Extensive Calibration Performed Prior to Return to JSC
  - Prototype Returned to JSC and Recalibrated on 6 July 1987
    - SwRI Informed JSC Accepts Repaired Unit

RAW DATA, SAPMD MODIFIED PROTOTYPE PERFORMANCE AT JSC JULY 6-8, 1987

185 F

	TOP TEMPERATURE			
	PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS
14.7	185	13.71		
14	183	13.00	181	13.00
12	185	11.01	183	11.01
10	186	9.02	183	8.95
8	187	7.03	184	6.96
6	187	5.05	185	5.05
4	186	3.06	185	3.06
2	187	1.14	181	1.14
0.03	185	0.00		

122 F

	TEMPERATURE INCREASING				TEMPERATURE DECREASING			
	PRESS DECREASING		PRESS INCREASING		PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS
14.7	122.2	14.71			121	14.57		
14	123.2	14.14	123.7	14.00	122	13.79	125	13.79
12	122.7	12.01	123.8	11.94	122	11.80	121	11.72
10	121.2	9.95	123.8	9.88	124	9.66	123	9.66
8	120.5	7.89	122	7.82	122	7.60	121	7.60
6	122.6	5.83	123.8	5.76	122	5.61	122	5.54
4	120.7	3.84	120.9	3.77	122	3.55	122	3.62
2	124	2.13	123.2	1.78	123	1.56	118	1.56
0.03	122.3	0.00			122	0.00		

95 F

	TEMPERATURE INCREASING				TEMPERATURE DECREASING			
	PRESS DECREASING		PRESS INCREASING		PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS
14.7	94.7	15.21			97	14.85		
14	96.6	14.35	95.8	14.28	97	14.07	93.7	14.14
12	95.4	12.36	93.8	12.22	93	12.01	96.6	12.01
10	95.7	10.16	94.4	10.16	93	10.02	94.2	9.81
8	96.7	8.17	97	8.17	96	7.82	95.5	7.82
6	94.7	6.11	94.3	6.04	97	5.83	94.1	5.83
4	93.8	4.05	95.7	3.98	96	3.77	93.9	3.77
2	96.3	2.13	93.4	1.92	95	1.71	95.8	1.71
0.03	94.5	0.00			96	0.00		

75 F

	INITIAL				TEMPERATURE INCREASING				FINAL			
	PRESSURE DECREASING		PRESSURE INCREASING		PRESS DECREASING		PRESS INCREASING		PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS
14.7	75.2	15.49			76.7	15.35			74.9	14.99		
14	76.5	14.57	77.0	14.57	73.3	14.64	74.7	14.64	75.2	14.21	74.4	14.28
12	75.4	12.44	76.2	12.51	73.3	12.58	74.0	12.51	76.0	12.22	74.7	12.22
10	76.4	10.45	76.6	10.37	74.0	10.45	74.0	10.37	74.2	10.09	76.6	10.09
8	74.5	8.31	75.3	8.31	73.3	8.39	75.3	8.39	75.7	7.53	76.6	8.03
6	75.7	6.25	77.0	6.32	73.9	6.25	74.7	6.25	76.4	5.90	74.6	5.97
4	76.5	4.12	73.5	4.12	75.3	4.12	74.0	4.12	73.6	1.78	74.5	1.78
2	73.5	2.06	73.4	2.06	74.6	2.13	76.0	2.13	75.8	0.00		
0.03	76.2	0.00			75.3	0.00						

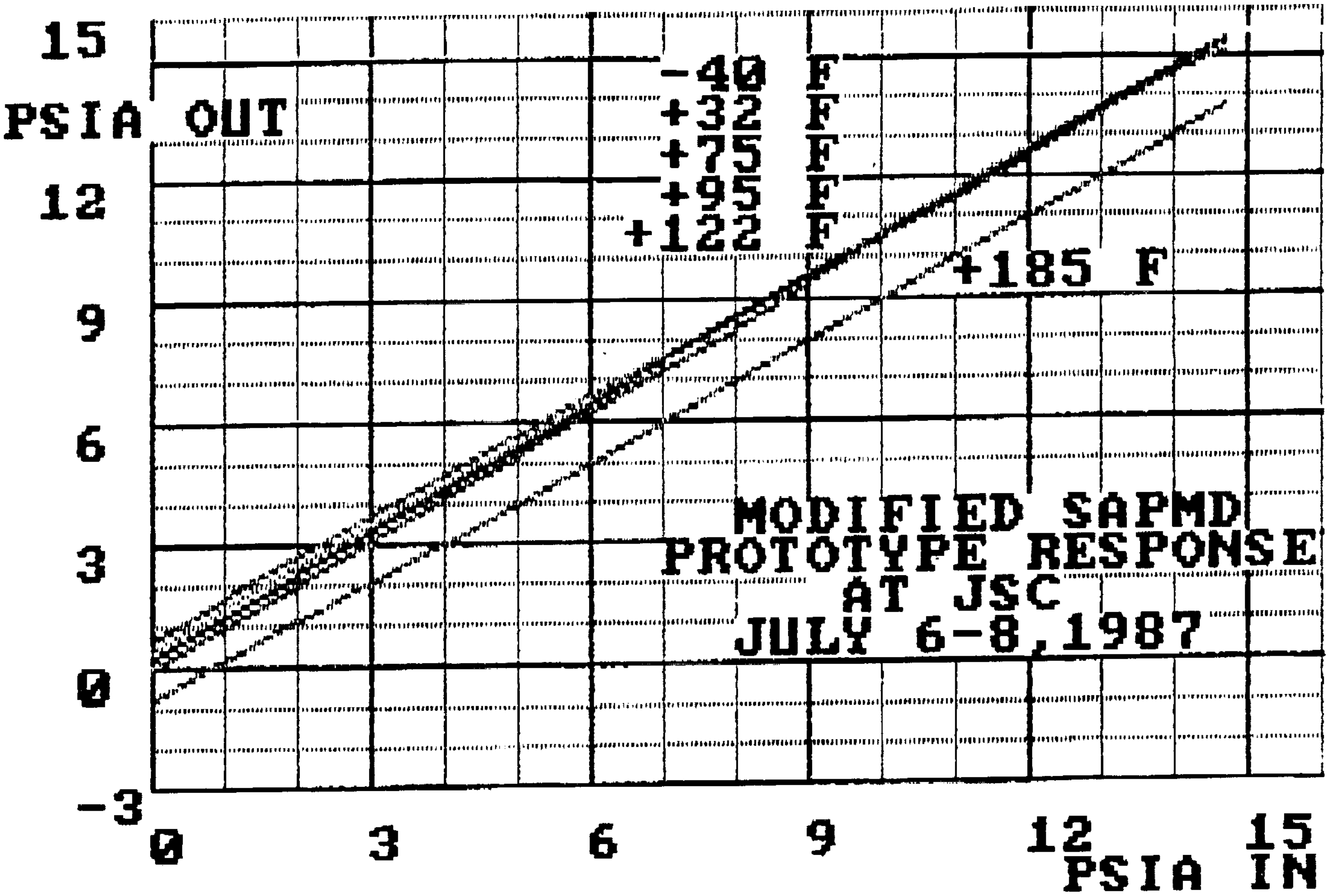
32 F

	TEMPERATURE DECREASING				TEMPERATURE INCREASING			
	PRESSURE DECREASING		PRESS INCREASING		PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS
14.7	32.7	15.35			32	15.63		
14	33.1	14.78	31.8	14.85	33	14.92	33.6	14.85
12	32.4	12.65	30.4	12.79	33	12.79	31.8	12.79
10	32.2	10.52	30.4	10.66	33.7	10.66	33.1	10.66
8	30.7	8.60	32.6	8.53	32.9	8.60	32.4	8.53
6	34.0	6.40	32.7	6.47	33.6	6.54	33.1	6.54
4	32.9	4.33	30.7	4.41	32.9	4.41	32.3	4.41
2	31.6	2.27	30.0	2.27	32.2	2.34	31.6	2.34
0.03	30.0	0.21			32.9	0.28		

RAW DATA, SAPMD MODIFIED PROTOTYPE PERFORMANCE AT JSC JULY 6-8, 1987

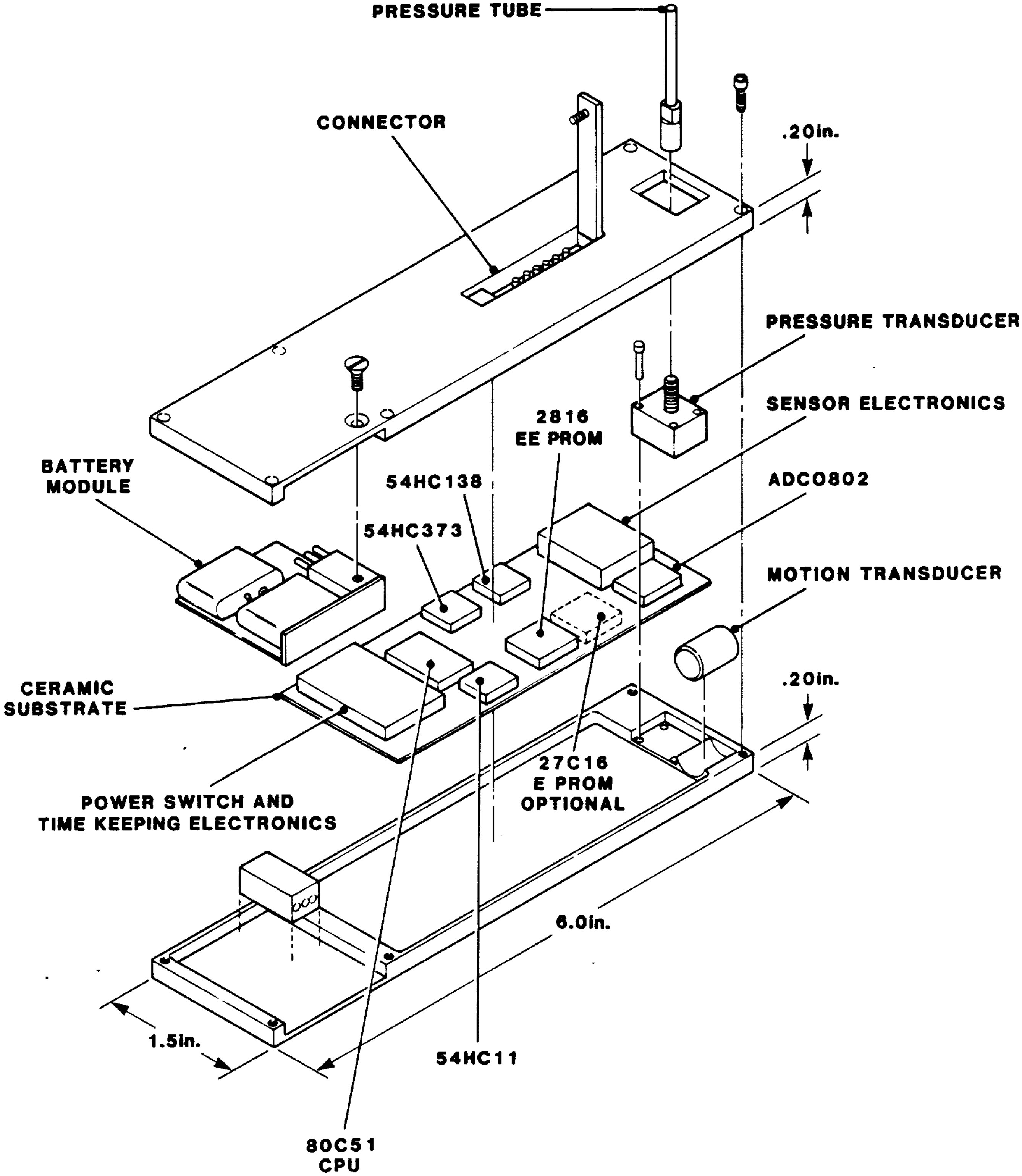
0 F	TEMPERATURE DECREASING				TEMPERATURE INCREASING			
	PRESS DECREASING		PRESS INCREASING		PRESS DECREASING		PRESS INCREASING	
	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS	TEMP	PRESS
14.7	1.1	15.49			1.0	15.63		
14	-1.9	14.78	0.1	14.78	-2.0	15.63	0	14.85
12	-1.9	12.79	0.1	12.72	-1.0	12.86	4	12.79
10	-0.3	10.66	-2.0	10.66	-2.0	10.80	4	10.66
8	0.4	8.60	0.4	8.67	-2.0	8.74	1	8.67
6	-0.3	6.54	-1.3	6.54	-1.0	6.68	0	6.68
4	-1.5	4.55	1.3	4.55	0.0	4.62	-2	4.55
2	1.3	2.49	0.6	2.49	-0.6	2.63	-2	2.56
0.03	1.0	0.43			-1.0	0.64		

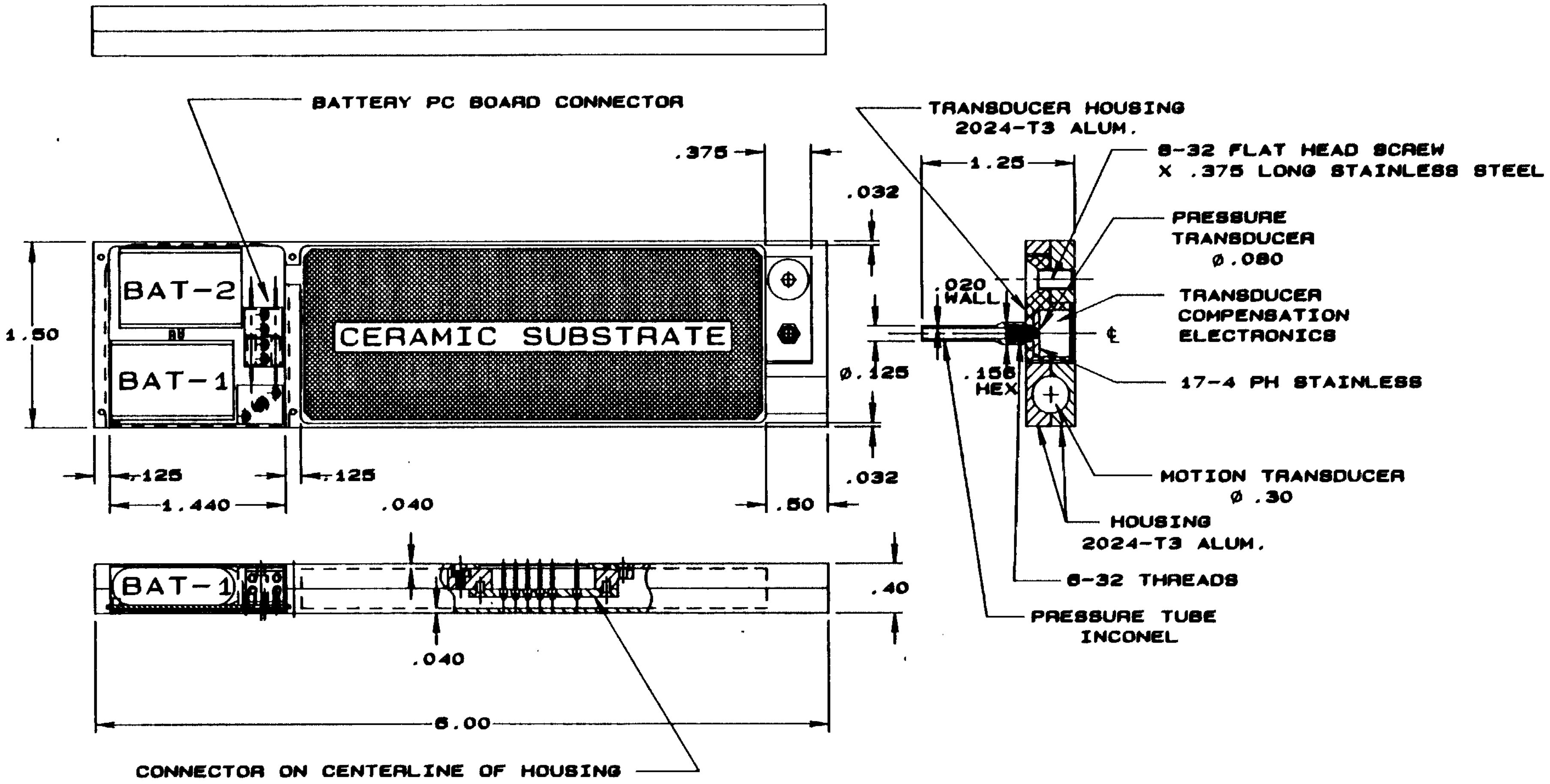
-40 F	BEFORE COLD SOAK		AFTER COLD SOAK	
	PRESSURE DECREASING		PRESSURE INCREASING	
	TEMP	PRESS	TEMP	PRESS
14.7	-41.2	15.14		
14	-41.3	14.57	-41	15.28
12	-41.5	12.58	-37	12.44
10	-39.2	10.59	-40	10.52
8	-38.5	8.53	-41.0	8.53
6	-41.6	6.68	-41.0	6.54
4	-40.0	4.69	-41.5	4.69
2	-42.0	2.70	-42.0	2.70
0.03	-40.0	0.64	-42.0	0.85


15
PSIA OUT
12
9
6
3
0
-3
0
3
6
9
12
15
PSIA IN
-40 F
+32 F
+75 F
+95 F
+122 F
+185 F
MODIFIED SAPMD
PROTOTYPE RESPONSE
AT JSC
JULY 6-8,1987

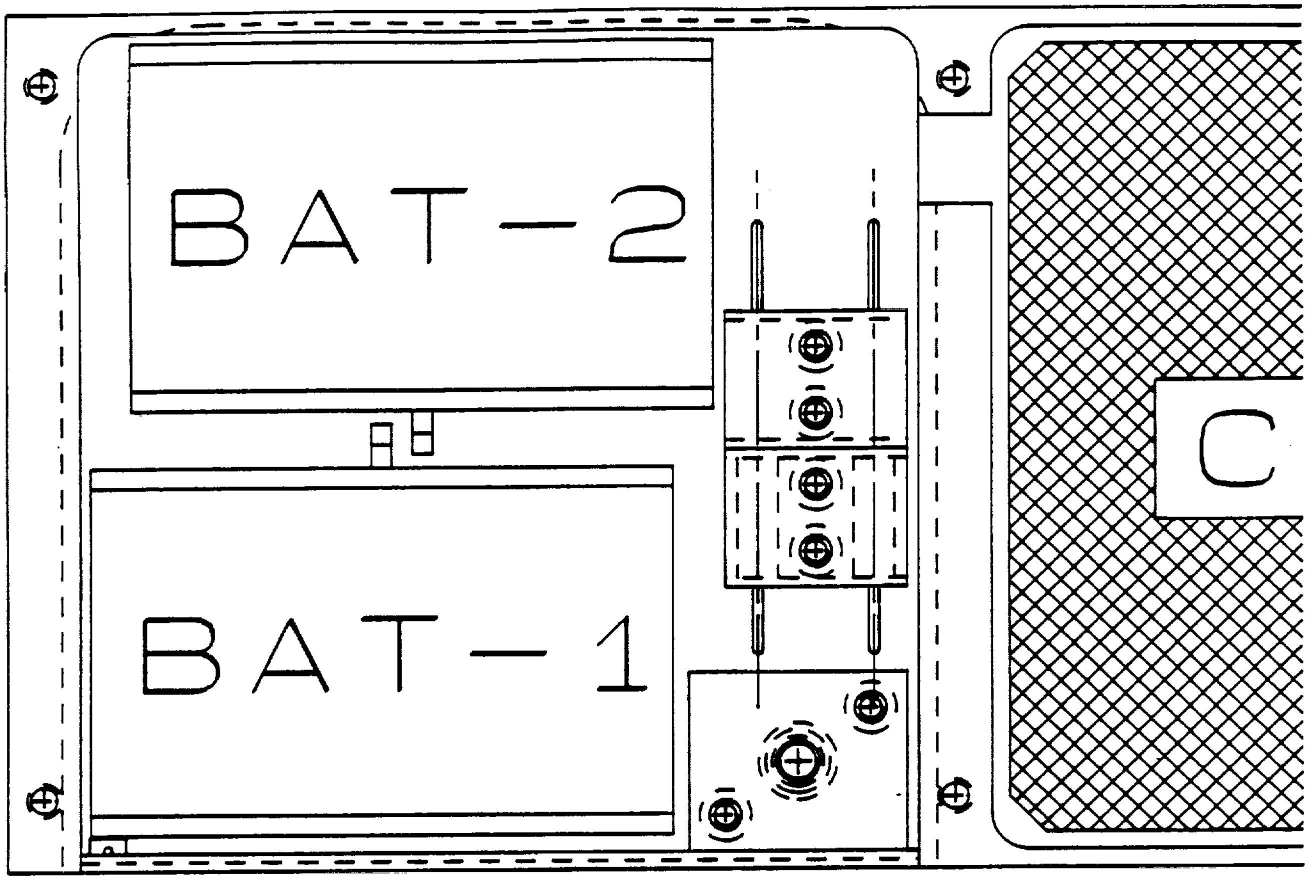
SAPMD ASSEMBLY DRAWING

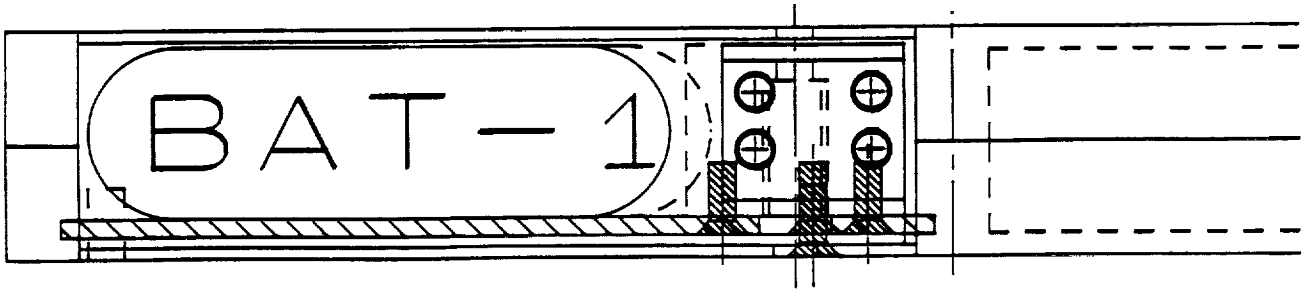
SAPMD ASSEMBLY

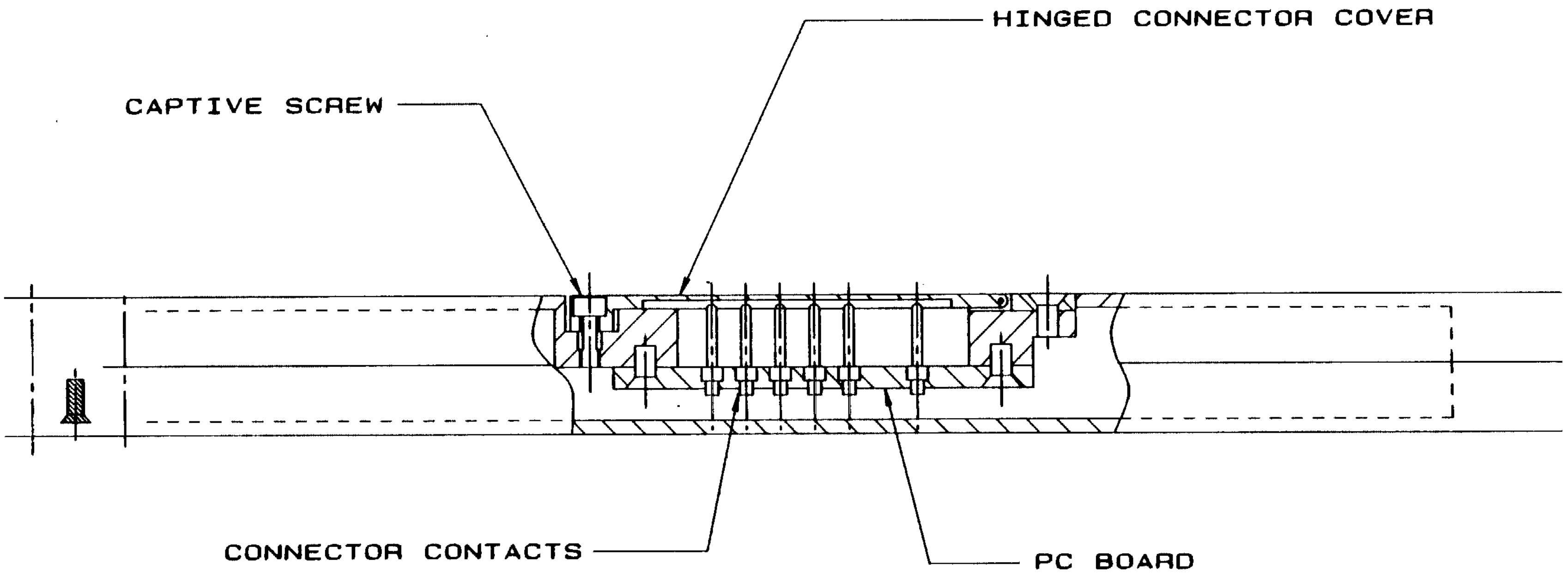



BATTERY PC BOARD CONNECTOR
TRANSDUCER HOUSING
2024-T3 ALUM.
6-32 FLAT HEAD SCREW
X .375 LONG STAINLESS STEEL
PRESSURE
TRANSDUCER
Ø .080
TRANSDUCER
COMPENSATION
ELECTRONICS
℄
17-4 PH STAINLESS
MOTION TRANSDUCER
Ø .30
HOUSING
2024-T3 ALUM.
6-32 THREADS
PRESSURE TUBE
INCONEL
.020
WALL
.156
HEX
1.25
.375
.032
Ø.125
.032
.50
1.50
BAT-2
BAT-1
CERAMIC SUBSTRATE
.125
.125
1.440
.040
.040
.40
6.00
CONNECTOR ON CENTERLINE OF HOUSING

SAPMD

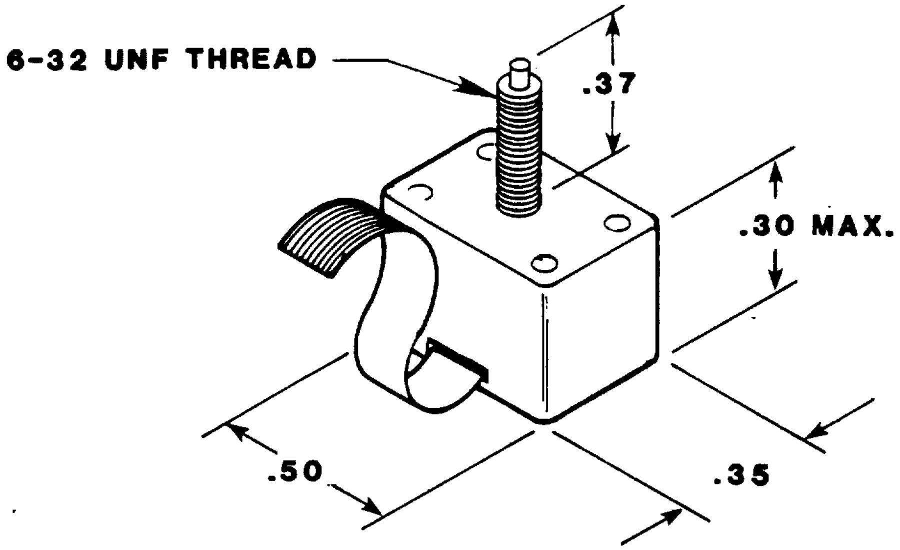

BAT-2
BAT-1
C


BAT-1

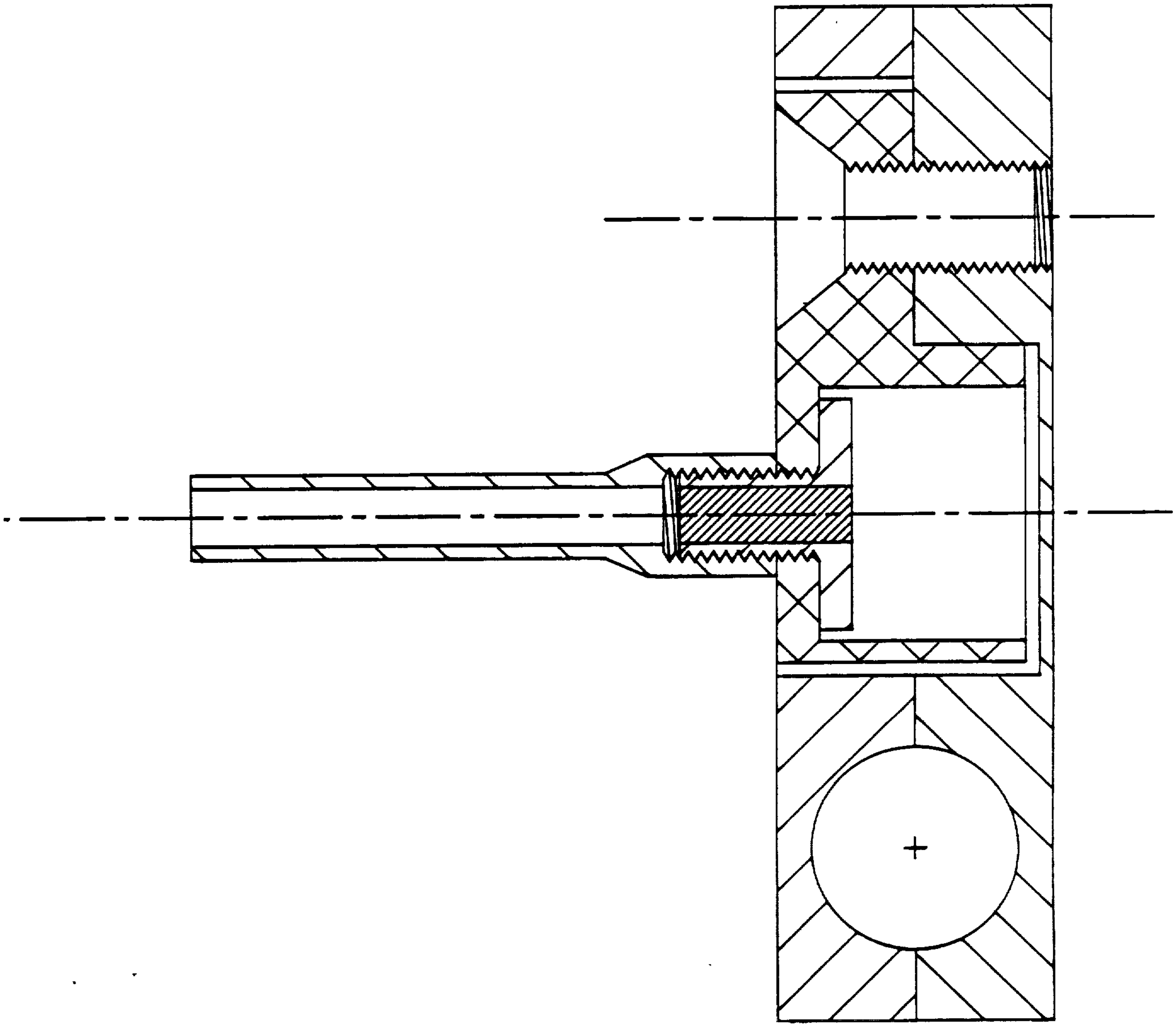


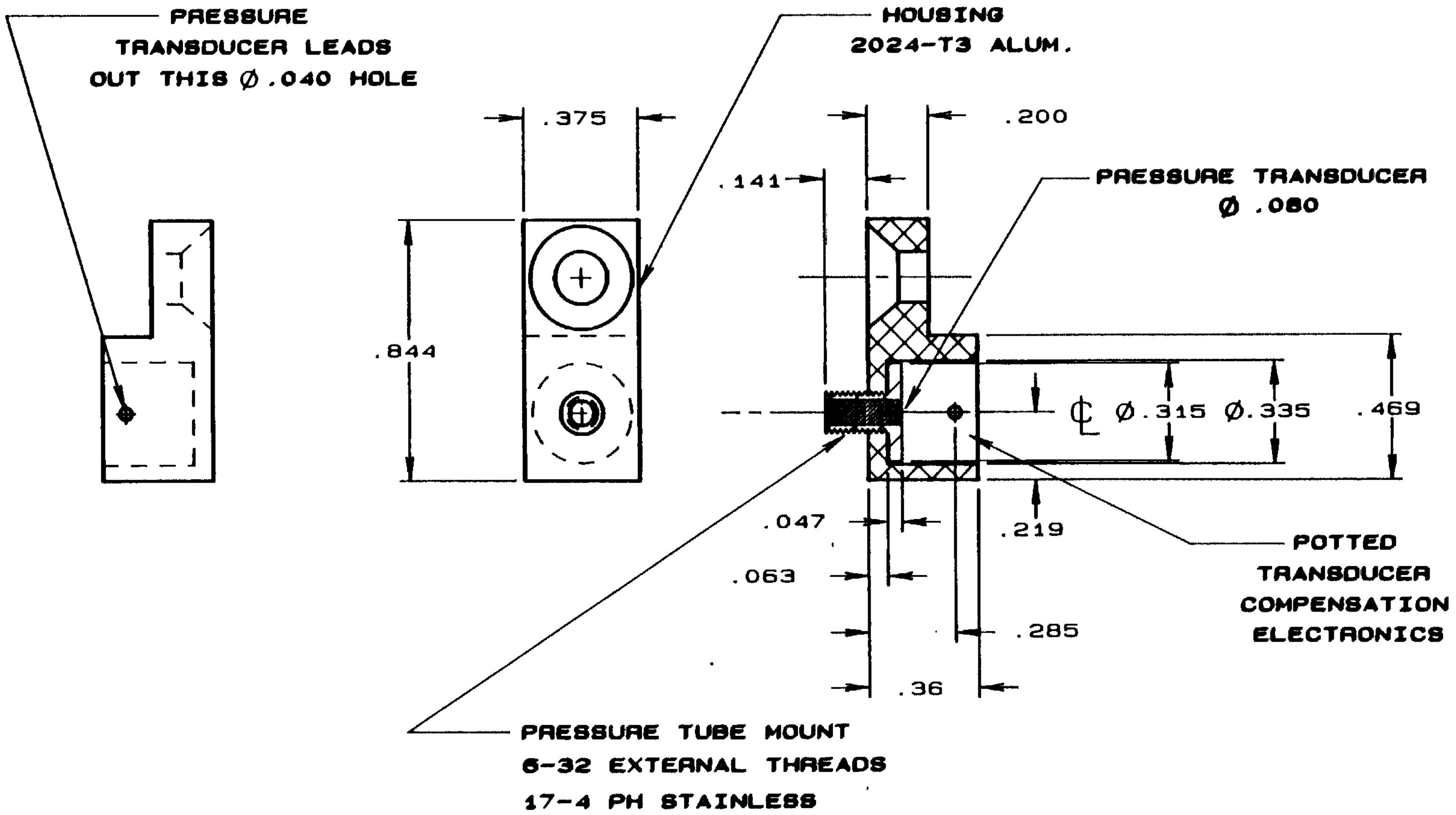
SAPMD CONNECTOR

PRESSURE TRANSDUCER

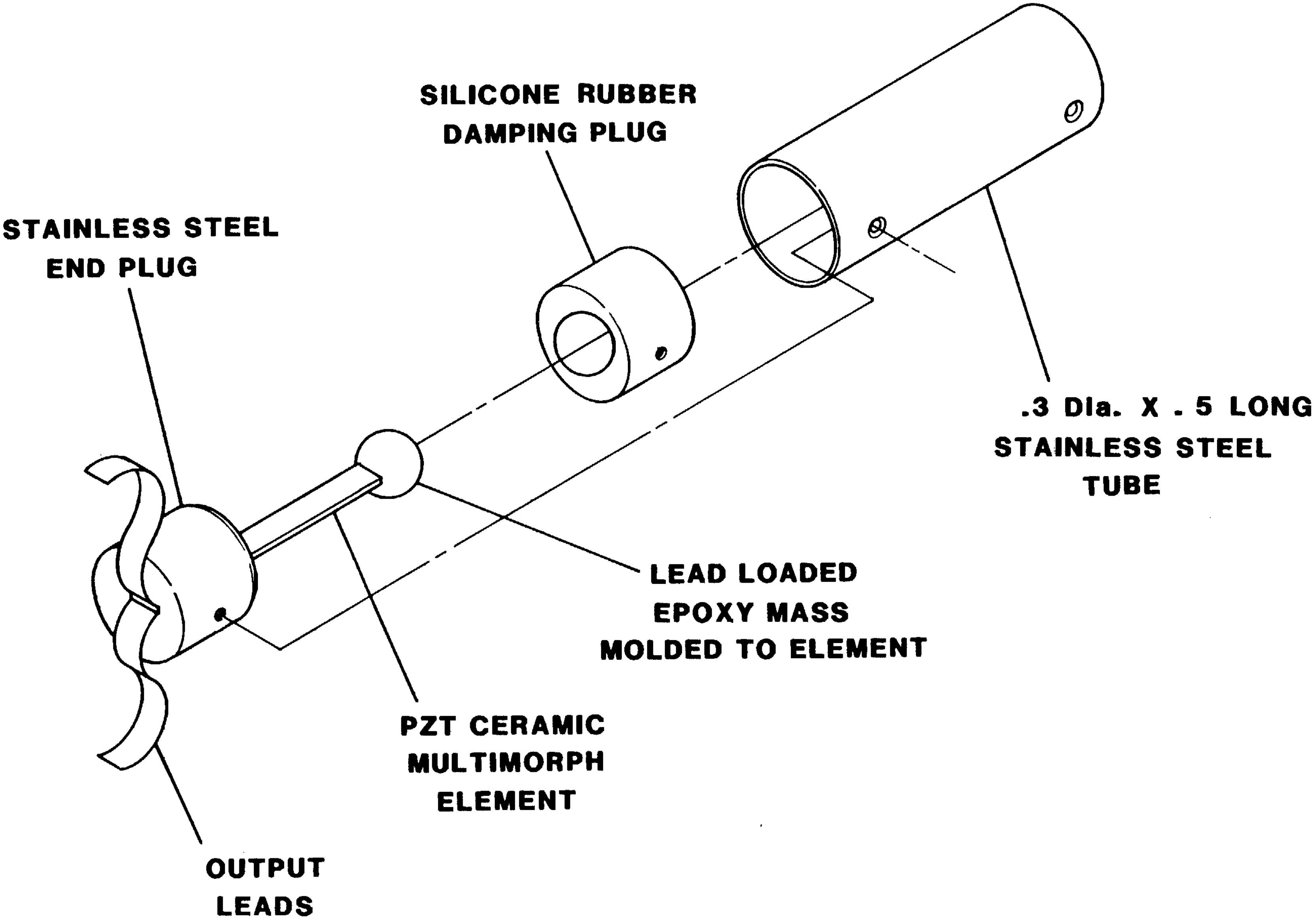


- ENTRAN EPI-080 TYPE PRESSURE TRANSDUCER
- CUSTOM PACKAGE
- INTERNAL COMPENSATION
- SILICON DIAPHRAM
- EXCITATION 5VDC
- STAINLESS STEEL HOUSING




PRESSURE
TRANSDUCER LEADS
OUT THIS Ø.040 HOLE
HOUSING
2024-T3 ALUM.
.375
.844
.200
.141
PRESSURE TRANSDUCER
Ø .080
℄ Ø.315 Ø.335
.469
.047
.219
.063
.285
.36
POTTED
TRANSDUCER
COMPENSATION
ELECTRONICS
PRESSURE TUBE MOUNT
6-32 EXTERNAL THREADS
17-4 PH STAINLESS

VIBRATION SENSOR


SILICONE RUBBER DAMPING PLUG
STAINLESS STEEL END PLUG
.3 Dla. X . 5 LONG STAINLESS STEEL TUBE
LEAD LOADED EPOXY MASS MOLDED TO ELEMENT
PZT CERAMIC MULTIMORPH ELEMENT
OUTPUT LEADS

FLIGHT SOFTWARE SUMMARY

SUMMARY OF
SAPMD FLIGHT SOFTWARE FUNCTIONS

* Detect Launch
* Log Pressure Data
* Hardware Self-Test
* Interract with GSE

SAPMD FLIGHT SOFTWARE MODULES

*	INIT	-	Check for Power-On or Tick Reset, Arm Watchdog Timer
*	TICK	-	Update Internal Clocks
*	Launch	-	Detect Launch and Record Pressure Sample
*	GSE	-	Communicate with SAPMD GSE
*	TIMEOUT	-	Suspend Operation of SAPMD Pending Reset
*	STATUS	-	Transmit Current SAPMD Status Over Serial Line
*	HANG	-	Enter 80C51 Power-Down Mode

SAPMD FLIGHT SOFTWARE EXECUTION LEVELS

* Critical Functions (High Priority)

Must Execute to Completion; Cannot be Suspended and Resumed or Interrupted by Reset.

INIT	-	Reset Test (Check Memory Bit Pattern)
TICK	-	Update Clocks
TIMEOUT	-	Suspend Operation

* Control Functions (Low Priority)

May be Interrupted by TIMEOUT, Suspended and Resumed

LAUNCH	-	Detect Launch, Record Pressure Sample
GSE	-	Communicate with GSE
STATUS	-	Transmit SAPMD Status

CONFIGURING SAPMD FLIGHT SOFTWARE

- SAPMD Operating Parameters in RAM/EEPROM may be Altered Using GSE

EEPROM Resident Parameters	RAM Resident Parameters
# Pressure Samples Recorded	Branch Points - Bail Out for S/W
First Free EEPROM Address	Clocks
EEPROM Size	Timeout Duration
SAPMD Serial Number	EEPROM Power Switch
	80C51 Special Function Resistor (SFR) Images
	Launch Detect Counter

SAPMD SELF-TESTS

- EEPROM Self-Test
  - Address Test
  - ALL 55H
  - ALL AAH
  - ALL FFH
  - ALL 00H
- Power System Test
- A/D Converter Test
- Pressure Transducer Test
- 80C51 RAM Test - Forces Power-On Reset
- 80C51 ROM Test
  - Compute and Compare with Recorded Checksum

PRESSURE DATA RECORDING FORMAT

- Up to 3 Acquisition Cycles in 2K EEPROM
- Each Sample Numbered
- Pressure File Format:

Byte	Content
0-3	GMT at Launch Detect
4-563	Pressure Samples

- Pressure Sample Format:

Byte	Content
0	Sequence #Mod 256
1	Pressure Transducer Reading

SAPMD STATUS INFORMATION

- Transmitted Every Tick
- Status Information:
  - EEPROM Powered
  - Self-Test in Progress
  - GSE Transaction in Progress
  - Data Acquisition in Progress
  - Acquisition Complete
  - Error
- RAM Dump:
  - RAM Block may be Dumped with Status to Monitor SAPMD Operation
- Status Information Displayed on GSE (GRID)

SGA SOFTWARE SUMMARY

SUMMARY OF
SAPMD GSE SHUTTLE GAUGE ACCESS (SGA) SOFTWARE

* Menu Driven

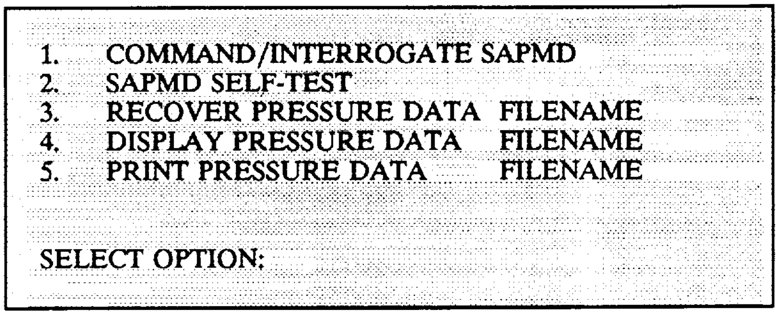
* Main Menu

* 2 Branch Menus

* Allows Interactive Access to SAPMD

* Provides SAPMD Calibration Offsets

SGA MAIN MENU

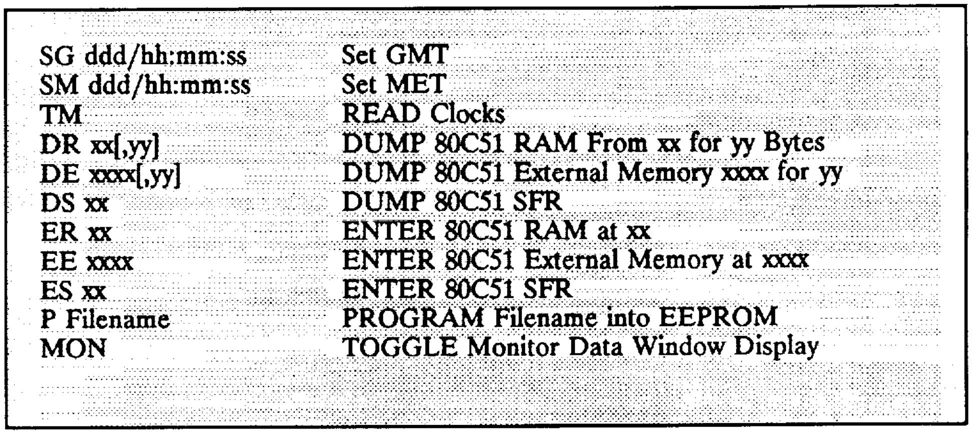


```
1.   COMMAND/INTERROGATE SAPMD
2.   SAPMD SELF-TEST
3.   RECOVER PRESSURE DATA  FILENAME
4.   DISPLAY PRESSURE DATA  FILENAME
5.   PRINT PRESSURE DATA    FILENAME


SELECT OPTION:
```

*   Options 1 and 2 use Branch Menus

OPTION 1: COMMAND/INTERROGATE SAPMD MENU

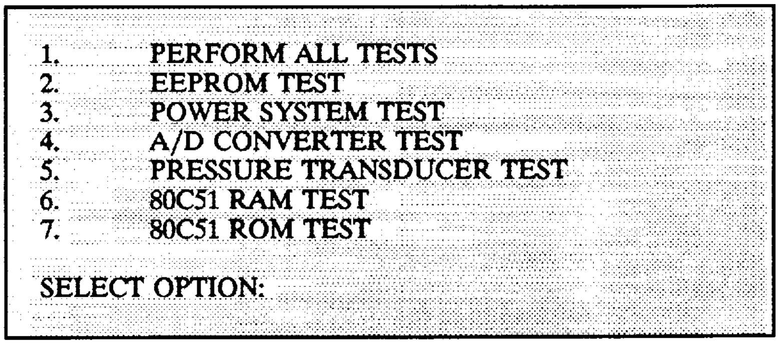


```
SG ddd/hh:mm:ss     Set GMT
SM ddd/hh:mm:ss     Set MET
TM                  READ Clocks
DR xx[,yy]          DUMP 80C51 RAM From xx for yy Bytes
DE xxxx[,yy]        DUMP 80C51 External Memory xxxx for yy
DS xx               DUMP 80C51 SFR
ER xx               ENTER 80C51 RAM at xx
EE xxxx             ENTER 80C51 External Memory at xxxx
ES xx               ENTER 80C51 SFR
P Filename          PROGRAM Filename into EEPROM
MON                 TOGGLE Monitor Data Window Display
```

>

* Commands are Entered Following ">" and Scroll Beneath Menu

OPTION 2: SAPMD SELF-TEST MENU



```
1.      PERFORM ALL TESTS
2.      EEPROM TEST
3.      POWER SYSTEM TEST
4.      A/D CONVERTER TEST
5.      PRESSURE TRANSDUCER TEST
6.      80C51 RAM TEST
7.      80C51 ROM TEST

SELECT OPTION:
```

* Test Executed and Completion Status Displayed by SGA

SAPMD STATUS DISPLAY

- Status Line:
  - EEPROM-On Self-Test GSE Acquisition Complete Error
- Displayed on Top Line of All Menu Screens
- Active Functions Indicated by Flashing Reverse Video
- Dumped RAM Displayed on Dedicated Window Which Replaces Command/Interrogate Menu
- "MON" Command Alternately Displays Monitored RAM Window or Command Menu

SAPMD<-->SGA DIALOG

*	All Commands and Responses ASCII

*	Command Character Reception Acknowledged by Echo

*	SGA or SAPMD May Abort Command at Any Time

*	All Commands Generate Completion Response

*	Status and Dumped RAM Data are Binary, Distinguished by Set High Order Bit of First Message Byte

SAPMD CALIBRATION

- Calibration Information
  - SAPMD Serial Number
  - Pressure Transducer Zero Bias
    - Number of Counts to Subtract from Each Sample to Adjust for Transducer Error
  - PSI/Count for Transducer Samples
- File is "SAPMD.CAL"

HP.SAPMD CEI SPECIFICATION

1.0 INTRODUCTION

This specification establishes the requirements specified in NAS9-17601, Request for Engineering Change Proposal dated April 19, 1988 for the design, development, fabrication, testing and delivery of two (2) Stand-Alone Pressure Measurement Devices (SAPMD) based on the design of the existing SAPMDs for the Shuttle Orbiter. The revised design will be capable of operation in a 1000 psi, 187°F environment for use in the Solid Rocket Booster (SRB) tests. The main tasks of this new work is the design of a high-pressure housing, substitution of a high-pressure sensor and design of a longer life battery supply.

1.1 Background

The present design of the SAPMD incorporates a microprocessor system implemented with hybrid module techniques using low power CMOS units which are contained in a 0.4-in. thick metal housing. The system is designed to operate installed under selected heat shield tiles on the Shuttle Orbiter and to thus survive and operate in a pressure regime from near zero psi to atmospheric pressure (14.7 psi). The unit operates from self-contained lithium batteries which provide an operating lifetime in the sleep mode of 1200 hours. The system is awakened by sensing the vibration of launch and takes pressure data for a period of 140 seconds with data readings every 100 ms. On-board real-time clock data are recorded with the pressure data. The data are recorded in EEPROMs which are capable of retaining the data indefinitely at temperatures up to 257°F. After retrieval of the module, data are supplied to GSE equipment for further use.

1.2 Specification Changes

In order to meet the requirements imposed by the SRB tests, the SAPMD must be redesigned to accommodate the new test environment. First, the housing of the SAPMD must be redesigned to survive a pressure of 1000 psi with at least a 50 percent overpressure capability. Protection of the internal electronic circuits from any mechanical stress is important for both reliability and accuracy. Second, a new pressure sensor must be selected which can measure pressures from 14.7 to 1000 psi with an accuracy of one percent FS over an ambient temperature range of 100 to 300°F. Third, the battery power supply must be modified to provide a lifetime of 2400 hours (100 days) of power-down operation. Forth, a new circuit must be added to allow an external hard-wired control line to activate the system prior to ignition. This circuit replaces the vibration sensor used to detect launch.

2.0 TECHNICAL APPROACH

The following paragraphs describe the technical work on the three tasks required to modify the SAPMD design for high pressure measurements.

2.1 Mechanical Design

The housing of the modified SAPMD shall be designed for operation at 1000 PSIA and 185°F. The housing of the monitor as shown in Figure 1 will contain the electronic circuits, batteries, and pressure transducer. The bottom cover of the housing is removable to provide access to the batteries and the data connector. The cover is sealed with an O-ring to insure that the internal pressure does not rise above 50 PSIA. The internal pressure must be limited since the batteries are sealed units and cannot withstand the high external ambient pressure. The housing structure shall be designed to withstand a maximum of pressure of 2000 PSIA to provide a 100% overpressure safety factor. If higher over-pressures are expected, the housing design can be modified either by the use of higher strength materials or by an increase in housing dimensions. The proposed design incorporates 300 series stainless steel as the housing material. The proposed design has no mounting holes assuming the unit will either be clamped or bonded in position. The design may be readily changed to provide attachment points if so required.

2.2 Electronic Design Changes

The changes in the electronic design of the SAPMD will be primarily in the start and stop command circuitry and, if necessary, to the bias circuit of the high-pressure transducer. The external start command will require additional circuitry included on the new battery board which will contain four (4) model LTC-7PN lithium batteries.

The start command will be controlled by a pair of wires connected to the high-pressure feed-thru of the SAPMD and routed outside the nozzle opening to a relay. This relay operation will be under control of the local firing-range officials and will be operated prior to ignition.

The contact closure activates a latch circuit inside the SAPMD which in turn enables a micro-powered voltage regulator to supply +5V to the SAPMD and commences recording pressure data. After a 160-second sampling period, the SAPMD returns to a power-down condition where it remains until turned back on by a command to the latch. This allows the system to be restarted in case of a misfire or delay.

The GSE will be used for interrogation, only, and will operate in the same manner as before, including supplying power to the SAPMD during interrogation.

While the SAPMD is in the power-down mode, the power drain will be less than 20 microamps of quiescent current of the regulator and latch circuitry.

2.3 Pressure Transducer

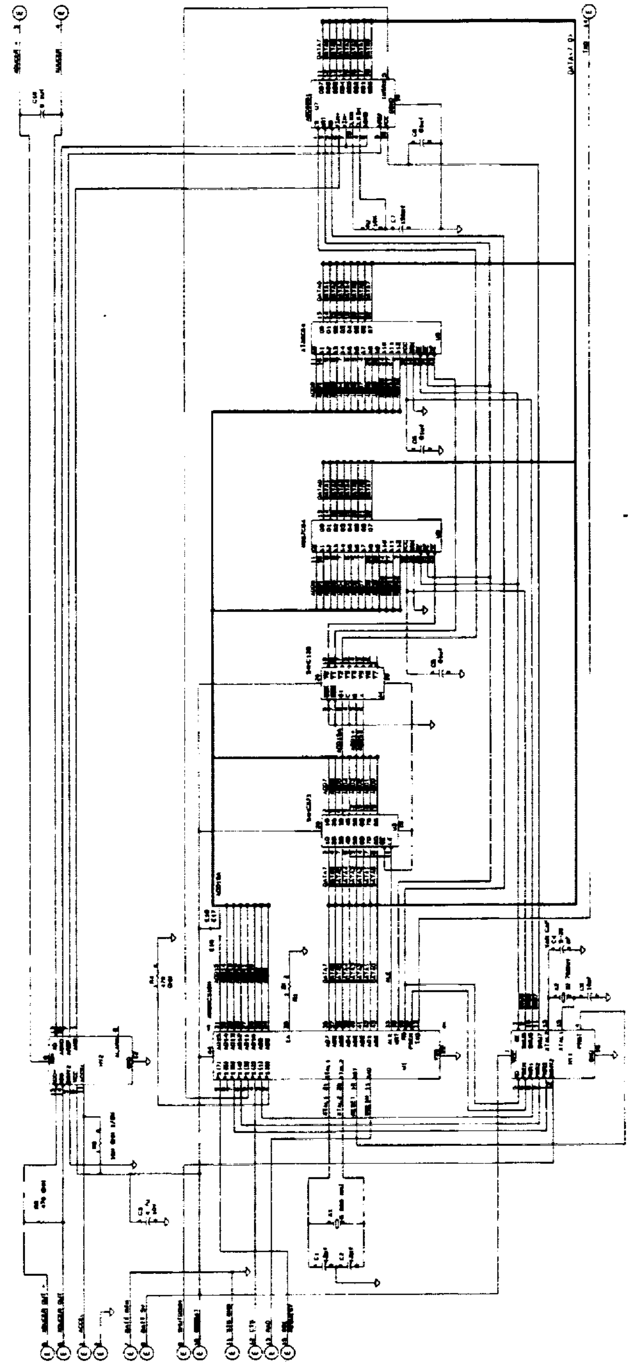
The pressure transducer selected is a standard Kulite XT-190 series ruggedized integrated sensor type absolute pressure transducer. The transducer is available with a maximum operating temperature of 350°F with a temperature compensated range of 100 to 300°F. The maximum change in sensitivity over the 100 to 300°F range is +/- 4.0 % with a repeatability of 0.05% of full scale with a 10 Vdc excitation voltage; the nominal output of the sensor is 100 mV full scale. The 8-bit resolution of the existing SAPMD A/D converter provides a theoretical resolution of 1000 psi/256 counts or 3.9 psi for the system. Actual measured performance of the present system indicates that a noise level of +/-3 LSB can be expected which gives a +/-11.7 psi noise level for the pressure measurements which combined with +/-3 psi non-linearity and hysteresis and 0.5 psi repeatability gives an error factor of +/-15.2 psi for single point measurements. The temperature coefficient of the transducer gives an expected error of +/-40 psi. The frequency response of this transducer will allow it to track the sum of the average pressure and the instantaneous acoustic pressure.

2.4 Battery Design

The required longer operational lifetime of the SAPMD and some additional power requirements by the pressure sensor will require more available battery power. The present battery power supply consists of two 3.4V lithium cells connected in series to supply a nominal 6.8V to a linear regulator which reduces the voltage to 4.5V for the electronic circuits. To increase the battery lifetime, four cells will be used in a series-parallel arrangement to supply 6.8V at twice the amp-hr rating of the existing supply and will be regulated down to 5V for the supply of the electronic circuits by a micro-powered regulator on the new battery board.

HP.SAPMD ELECTRONIC SCHEMATICS

ORIGINAL PAGE IS
OF POOR QUALITY



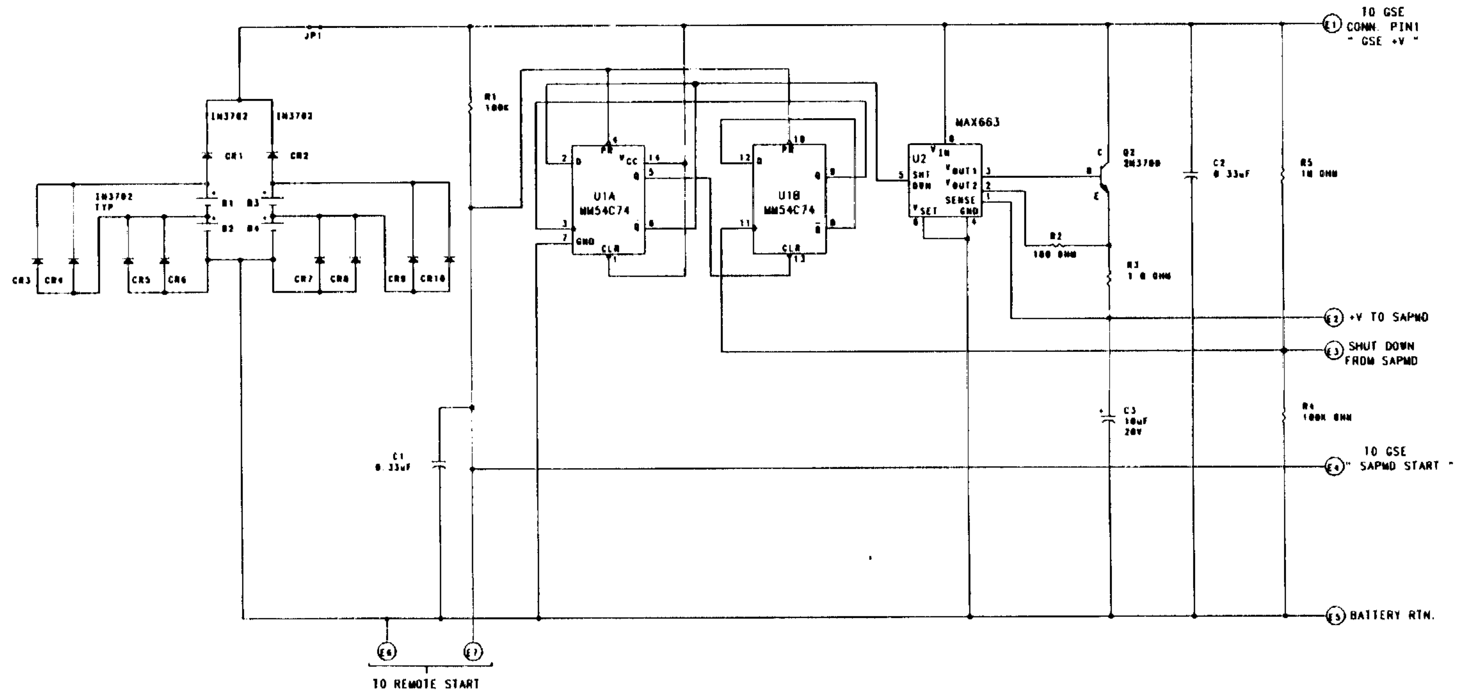
PARTS LIST

TOLERANCES

SOUTHWEST RESEARCH INSTITUTE

D 26401

REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED



APPROVALS SIGNATURE & DATE		
DRAWN	KA McKee	3/88
CHECKED		
ELECT		
QA		
PROJECT		

SOUTHWEST RESEARCH INSTITUTE
SAN ANTONIO TEXAS

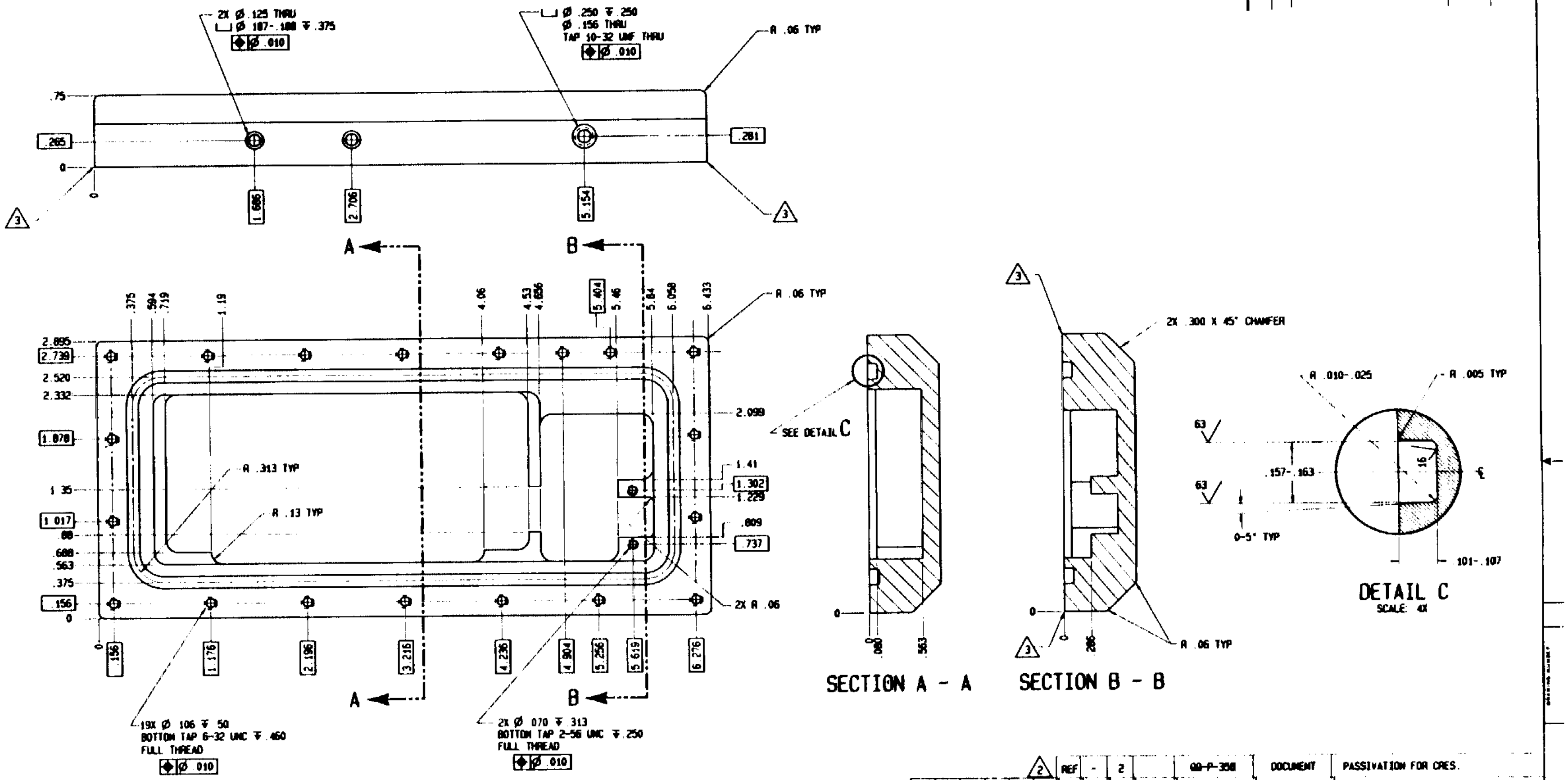
SCHEMATIC, BATTERY BOARD
HP / SAPMD

CONTRACT

SIZE	CODE IDENT NO	DRAWING NO
C	26401	15-1062-127

SCHEMATIC SHEET 1 OF 1

HP.SAPMD MECHANICAL SCHEMATICS



NOTES:

1. BREAK ALL SHARP EDGES AND CORNERS .020 MAX, EXCEPT WHERE NOTED.

△2 PASSIVATE ENTIRE PART PER FIND NO. 2.

△3 DO NOT BREAK SHARP EDGE.

QTY	UNIT OF MEAS	FIND NO	CODE IDENT NO	PART OR IDENTIFYING NO	NOMENCLATURE OR DESCRIPTION	NOTE TEXT
REF	-	2		QQ-P-35B	DOCUMENT	PASSIVATION FOR CRES.
1	EA	1			BASE	17-4 PH S. STEEL, PER AMS-5643

PARTS LIST

TOLERANCES

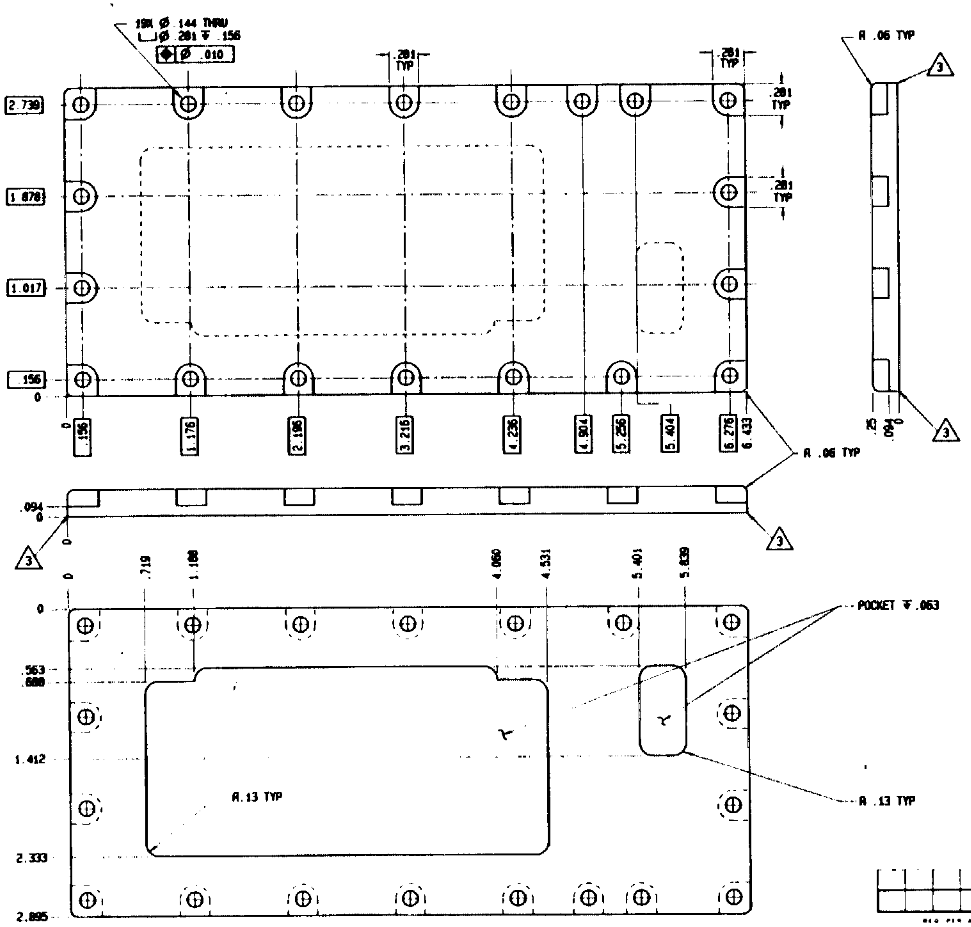
SOUTHWEST RESEARCH INSTITUTE

HOUSING, HP-SAPMD

SIZE D CODE IDENT NO 26401 DRAWING NO 15-1062-202

SHEET 1 OF 1

ORIGINAL PAGE IS
OF POOR QUALITY



ORIGINAL PAGE IS
OF POOR QUALITY

NOTES:
1. BREAK ALL SHARP EDGES AND CORNERS .020 MAX. EXCEPT WHERE NOTED.
2. PASSIVATE ENTIRE PART PER FIND NO. 2.
3. DO NOT BREAK ALL SHARP EDGE.

QTY REQ PER ASSY INST	UNIT OF MEAS	FIND NO	CODE IDENT NO	PART OR IDENTIFYING NO	NOMENCLATURE OR DESCRIPTION	STOCK SIZE MATERIAL SPECIFICATION TYPE CLASS GRADE HEAT TREAT NOTE TEXT
REF	-	2		QQ-P-35B	DOCUMENT	PASSIVATION FOR CRES.
1	EA	1			BASE	17-4 PH S. STEEL, PER AMS-5643

PARTS LIST

NEXT ASSY | USED ON
APPLICATION

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES AND INCLUDE CHEMICALLY APPLIED OR PLATED FINISHES

TOLERANCES

CONTRACT
DES D. HEJDUK 8/24/88
DRN A. RAMIREZ 8/26/88
APPROVALS SIGNATURE & DATE
CHK
MECH W R P 8/26/88
ELEC
PROJ

SOUTHWEST RESEARCH INSTITUTE
SAN ANTONIO, TEXAS

BASE, HP-SAPMD

SIZE D | CODE IDENT NO 26401 | DRAWING NO 15-1062-203
SCALE 1 : 1 | SHEET 1 OF 1

HP.SAPMD GSE SOFTWARE LISTINGS

PACKING.LST   January 3, 1989

The following files are include with this distribution:

```
HPSGA     EXE     72520   1-03-89    5:06p    Executable Program

README    HP       2765   1-03-89    5:53p    Program Notes
PACKING   LST      2198   1-03-89    6:23p    This file

HPSGA              1284   1-03-89    3:47p    MAKE file for HPSGA

HPSGALNK  LNK       121   1-03-89    3:48p    Link command file for HPSGA

CMDINT    C       30491  12-19-88    6:09p    C Source Files
DBCMD     C        9472   1-01-80   12:06a
DEBUG     C        3468   1-03-89    3:50p
DIPRESS   C        6561   1-03-89    4:00p
ERROR     C        2745  12-28-88   12:32p
HPSGA     C         264  12-28-88    1:59p
MENU      C        3860   1-03-89    5:05p
PRPRESS   C        2800   1-03-89    4:07p
RECOVER   C        4122  12-29-88    4:29p
SELFTEST  C        4332   1-03-89    5:03p
STATUS    C        2359   1-03-89    3:51p
SUPGLOB   C        9874   1-03-89    4:04p
WINDOW    C       11136   1-01-80    4:05p

CONIO     ASM      2304   1-01-80   10:36a    Assembly Source Files
FIO       ASM      8320   1-01-80    3:17p
_WINDOW   ASM      7061  12-19-88    6:25p

CONIO     LST      5501  12-19-88    6:26p    Assembly List Files
FIO       LST     17655  12-19-88    6:26p
_WINDOW   LST     16591  12-19-88    6:25p

CONIO     CRF       749  12-19-88    6:26p    Assembly Cross Reference Files
FIO       CRF      2937  12-19-88    6:26p
_WINDOW   CRF      2378  12-19-88    6:25p

HPSGA     MAP     17359   1-03-89    5:06p    Link map for HPSGA

CMDINT    OBJ      5872   1-03-89    4:09p    Object Modules
CONIO     OBJ       167  12-19-88    6:26p
DBCMD     OBJ      2242   1-03-89    4:06p
DEBUG     OBJ      1475   1-03-89    4:09p
DIPRESS   OBJ      2297   1-03-89    4:07p
ERROR     OBJ      1261   1-03-89    4:05p
FIO       OBJ       790  12-19-88    6:26p
HPSGA     OBJ       965   1-03-89    4:05p
MENU      OBJ      2036   1-03-89    5:05p
PRPRESS   OBJ      1636   1-03-89    4:07p
RECOVER   OBJ      1203   1-03-89    4:07p
SELFTEST  OBJ      1654   1-03-89    5:05p
STATUS    OBJ      1194   1-03-89    4:07p
WINDOW    OBJ      2261   1-03-89    4:05p
_WINDOW   OBJ       639  12-19-88    6:25p
```

```
# make file for hpsga
supglob.c : \include\stdio.h \include\process.h \include\stdlib.h
    m supglob.c     #allow change to supglob so that all else will compile

hpsga.obj : hpsga.c supglob.c
    cl /I\sapmd hpsga.c /c

window.obj : window.c supglob.c
    cl /I\sapmd window.c /c

error.obj : error.c supglob.c
    cl /I\sapmd error.c /c

dbcmd.obj : dbcmd.c supglob.c
    cl /I\sapmd dbcmd.c /c

menu.obj : menu.c supglob.c
    cl /I\sapmd menu.c /c

prpress.obj : prpress.c supglob.c
    cl /I\sapmd prpress.c /c

status.obj : status.c supglob.c
    cl /I\sapmd status.c /c

dipress.obj : dipress.c supglob.c
    cl /I\sapmd dipress.c /c

recover.obj : recover.c supglob.c
    cl /I\sapmd recover.c /c

selftest.obj : selftest.c supglob.c
    cl /I\sapmd selftest.c /c

cmdint.obj : cmdint.c supglob.c
    cl /I\sapmd cmdint.c /c

debug.obj : debug.c supglob.c
    cl /I\sapmd debug.c /c

_window.obj : _window.asm
    masm _window.asm,,,;

conio.obj : conio.asm
    masm conio.asm,,,;

fio.obj : fio.asm
    masm fio.asm,,,;

hpsga.exe : hpsga.obj window.obj _window.obj error.obj dbcmd.obj conio.obj \
fio.obj menu.obj prpress.obj status.obj dipress.obj recover.obj selftest.obj \
cmdint.obj debug.obj
    link @hpsgalnk.lnk
```

```
/********************************************************************************************/
/*                                                                                          */
/*                                    C M D I N T                                           */
/*                                                                                          */
/*       Command and interrogate the SAPMD.                                                 */
/*                                                                                          */
/***********************************************************                                */
                                                         /*                                 */
#include <supglob.c>                                     /* locate global data              */
                                                         /*                                 */
cmdint()                                                 /* command/interrogate SAPMD */
     {int day,                                           /* date day                        */
           hh,                                           /* hh:mm:ss                        */
           mmss[2],                                      /* ...                             */
           cmd,                                          /* command                         */
           i;                                            /* iteration variable              */
      unsigned char adrct[3];                            /* address and count               */
      unsigned char *tptr;                               /* time pointer                    */
      unsigned long ticks;                               /* ...                             */
      screen.lines++;                                    /* add line for bottom             */
      screen.cury++;                                     /* ...                             */
      menu(2,1);                                         /* display menu                    */
      while (1)                                          /* loop forever ...                */
          {wchs(CR);                                     /* new line                        */
           prompt(">");                                  /* type prompt character           */
           if ((i=rdln())==LEFT || i==HOME)              /* return to top menu?             */
              {screen.lines--;                           /* restore screen size ...         */
               screen.cury--;                            /* ...                             */
               return(0);};                              /* ---> return                     */
           switch (scan())                               /* look for command                */
                                                         /*                                 */
/***********************************************************                                */
/*                                                                                          */
/*        SG, SM: Set GMT, MET                                                              */
/*                                                                                          */
/***********************************************************                                */
                                                         /*                                 */
               {case SG:                                 /* set-gmt                         */
                    cmd=SETGMT;                          /* establish SAPMD command         */
                    goto l1;                             /*                                 */
                case SM:                                 /* set-met                         */
                    cmd=SETMET;                          /* establish command               */
l1:                                                      /*                                 */
                   if (scan()==NUMBER && (day=acc)<=365) /* get day ...                     */
                        if (scan()==SLASH)               /* eat '/'                         */
                             if (scan()==NUMBER && (hh=acc)<24) /* get hour ...             */
                                  {for (i=0;i<2;i++)     /* get minutes and seconds         */
                                       {if (scan()==COLON) /* get ':'                       */
                                            if (scan()==NUMBER && (mmss[i]=acc)<60)
                                                 continue; /* next iteration                */
                                        break;};         /* exit loop on error              */
                                   if (i>=2)             /* :hh:ss present?                 */
                                       if (scan()==EOL)  /* good terminator?                */
                                            {ticks=day*0x2a3001+(unsigned long)hh*7200+
                                                     mmss[0]*120+mmss[1]*2; /* 1/2 secs*/
                                             if (!sacmd(cmd,&ticks,4) || rdsg()!=ACK)
                                                 p_error(BADSAPMD); /* bad response         */
                                             break;}};   /* next command                    */
                   error(BADCMD);                        /* strange command                 */
                   break;                                /* next                            */
                                                         /*                                 */
/***********************************************************                                */
/*                                                                                          */
/*        TM: read GMT, MET                                                                 */
/*                                                                                          */
/***********************************************************                                */
```

```
                                                /*                           */
             case TM:                           /* read GMT                  */
                if (scan()==EOL)                /* check for good command    */
                    {adrct[0]=GMTADR;           /* point to memory           */
                     adrct[1]=8;                /* send byte count           */
                     if (!sacmd(DUMPRAM,adrct,2) || versg(RAMDATA)) /* issue */
                        {p_error(BADSAPMD);     /* strange response          */
                         break;};               /* next command              */
                     rdtime();};                /* read and display times    */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                            */
/*                                                                           */
/*      DR: Dump 80C51 ram                                                   */
/*                                                                           */
/*************************************************                            */
                                                /*                           */
             case DR:                           /* dump 80C51 ram            */
                schex();                        /* get address               */
                adrct[0]=acc;                   /* save parameter            */
                if (acc<=0x7f)                  /* check range               */
                    {adrct[1]=16;               /* default byte count        */
                     if (scan()==COMMA)         /* check for count           */
                        {if (scan()==NUMBER)    /* a number there?           */
                            {adrct[1]=acc & 0xff; /* make good number        */
                             scan();}           /* get EOL                   */
                         else                   /* no number                 */
                             {error(BADCMD);    /* strange command           */
                              break;}};         /* next                      */
                     if (token==EOL)            /* check for garbage         */
                        {if (!sacmd(DUMPRAM,adrct,2) || versg(RAMDATA))
                            p_error(BADSAPMD);  /* print error               */
                         else                   /* command ok                */
                            dump(adrct[0],adrct[1]);/* read and print resp.  */
                         break;}};              /* next                      */
                error(BADCMD);                  /* strange command           */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                            */
/*                                                                           */
/*      DS: Dump 80C51 SFR                                                   */
/*                                                                           */
/*************************************************                            */
                                                /*                           */
             case DS:                           /* dump SFR                  */
                if (schex()==NUMBER && (adrct[0]=acc)>0x7f && acc<=255) /*   */
                    if (scan()==EOL)            /* good command?             */
                        {if (!sacmd(DUMPSFR,adrct,1) || versg(SFRDATA))
                            p_error(BADSAPMD);  /* send error                */
                         else                   /*                           */
                            {adrct[1]=1;        /* fake byte count           */
                             dump(adrct[0],adrct[1]); /* display SFR contents*/
                             break;};           /* next                      */
                error(BADCMD);                  /* strange command           */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                            */
/*                                                                           */
/*      DE: Dump external memory                                             */
/*                                                                           */
/*************************************************                            */
                                                /*                           */
             case DE:                           /* dump 80C51 ram            */
                schex();                        /* get address               */
                adrct[0]=acc;                   /* save parameter            */
                adrct[1]=acc>>8;                /* ...                       */
                adrct[2]=16;                    /* default byte count       */
                if (scan()==COMMA)              /* check for count          */
                    {if (scan()==NUMBER)        /* a number there?          */
                        {adrct[2]=acc & 0xff;   /* make good number         */
                         scan();}               /* get EOL                  */
                     else                       /* no number                */
                         {error(BADCMD);        /* strange command          */
                          break;}};             /* next                     */
                 if (token==EOL)                /* check for garbage        */
                    {if (!sacmd(DUMPEXT,adrct,3) || versg(EXTDATA))
                        p_error(BADSAPMD);      /* print error              */
                     else                       /* command ok               */
                        dump(adrct[1]<<8|adrct[0],adrct[2]); /* read, print */
                     break;}};                  /* next                     */
                error(BADCMD);                  /* strange command          */
                break;                          /* next                     */
                                                /*                          */
/*************************************************                          */
/*                                                                          */
/*      ER: Enter 80C51 ram                                                 */
/*                                                                          */
/*************************************************                          */
                                                /*                          */
             case ER:                           /* enter ram                */
                schex();                        /* get address              */
                i=acc;                          /* save address             */
                if (i<=127)                     /* check for good address   */
                    if (scan()==EOL)            /* check for good command   */
                        {enter(DUMPRAM,LOADRAM,i,0x7f,1,RAMDATA); /* ram    */
                         break;}                /* next                     */
                error(BADCMD);                  /* strange command          */
                break;                          /* next                     */
                                                /*                          */
/*************************************************                          */
/*                                                                          */
/*      ES: Enter 80C51 SFR                                                 */
/*                                                                          */
/*************************************************                          */
                                                /*                          */
             case ES:                           /* enter ram                */
                schex();                        /* get address              */
                i=acc;                          /* save address             */
                if (i<=255 && i>=128)           /* check for good address   */
                    if (scan()==EOL)            /* check for good command   */
                        {enter(DUMPSFR,LOADSFR,i,0x7f,1,SFRDATA); /* load SFR*/
                         break;}                /* next                     */
                error(BADCMD);                  /* strange command          */
                break;                          /* next                     */
                                                /*                          */
/*************************************************                          */
/*                                                                          */
/*      EE: Enter EEPROM                                                    */
/*                                                                          */
/*************************************************                          */
                                                /*                          */
             case EE:                           /* enter ram                */
                schex();                        /* get address              */
                i=acc;                          /* save address             */
                if (scan()==EOL)                /* check for good command   */
                    {enter(DUMPEXT,LOADEE,i,0xffff,2,EXTDATA); /* load EEPROM*/
                     break;};                   /* next                     */
                error(BADCMD);                  /* strange command          */
                break;                          /* next                     */
                                                /*                          */
/*************************************************                          */
/*                                                                          */
/*      P: Program file into EEPROM                                          */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
             case P:                            /* program file              */
                if (i=prog()) error(i);         /* read command              */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*      MON: Toggle ram display window                                       */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
             case MON:                          /* toggle monitor window     */
                if (m5->disp)                   /* monitor window displayed? */
                    menu(2,0);                  /* restore menu              */
                else                            /* menu displayed            */
                    menu(4,0);                  /* display monitor window    */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*      CMD: Execute command file                                            */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
             case CMD:                          /* execute command file      */
                if (i=excfile()) error(i);      /* open command file         */
                break;                          /* next                      */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*      LA: Set up for rubber launch                                         */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
             case LA:                           /* rubber launch             */
                if (scan()==NUMBER && (day=acc)<=365) /* get day ...         */
                    if (scan()==SLASH)          /* eat '/'                   */
                        if (scan()==NUMBER && (hh=acc)<24) /* get hour ...   */
                            {for (i=0;i<2;i++)  /* get minutes and seconds   */
                                {if (scan()==COLON) /* get ':'               */
                                    if (scan()==NUMBER && (mmss[i]=acc)<60)
                                        continue; /* next iteration          */
                                 break;};       /* exit loop on error        */
                             if (i>=2)          /* :hh:ss present?           */
                                if (scan()==EOL) /* good terminator?         */
                                    {ticks=day*0x2a3001+(unsigned long)hh*7200+
                                           mmss[0]*120+mmss[1]*2; /* 1/2 secs*/
                                           if ((cfile=fopen("la.cmd","rb"))==NUL
                                                {error(NOLAFILE);/* no la.cmd*/
                                                 break;} /* bail out         */
                                           else /* good open                 */
                                                cmdfile=1; /* flag cmd file */
                                           tptr=(char *) &ticks; /* point to gmt
                                           adrct[0]=0;  /* make address      */
                                           adrct[1]=0xb0; /* ...             */
                                           for (i=0;i<4;i++) /* plant gmt    */
                                                {adrct[2]=*tptr++; /* get tim*/
                                           if (!sacmd(LOADEE,adrct,3) || rdsg()!
                                                {p_error(BADSAPMD);
                                                 goto 12;}; /* bail out      */
                                                 adrct[0]++;};/* next byte   */
                                     break;}};  /* next command              */
                error(BADCMD);                  /* strange command           */
```

```
                                                /*                            */
/*************************************************                            */
/*                                                                            */
/*                              R P B Y T E                                   */
/*                                                                            */
/*      Read and display byte.  Check for errors.                             */
/*                                                                            */
/*************************************************                            */
                                                /*                            */
rpbyte()                                        /* read and print byte        */
    {unsigned char rch;                         /* character read             */
     int i;                                     /* iteration variable         */
     for (i=0;i<2;i++)                          /* read 2 bytes               */
        {if ((rch=rdsg())==ABORT)               /* stop?                      */
            {p_error(BADSAPMD);                 /* strange response           */
             return(0);};                       /* bail out                   */
         wchs(rch);}                            /* display character          */
     return(1);}                                /* ---> return                */
                                                /*                            */
/*************************************************                            */
/*                                                                            */
/*                              R D T I M E                                   */
/*                                                                            */
/*      Read and display GMT and MET.                                         */
/*                                                                            */
/*************************************************                            */
                                                /*                            */
rdtime()                                        /* read times                 */
    {static char *gmetxt[]={"GMT: ","MET: "};   /* time text                  */
     int i,                                     /* iteration variable         */
         j,                                     /* iteration variable         */
         k,                                     /* iteration variable         */
         jtime[5];                              /* GMT                        */
     union {unsigned char byt[4];               /* time value                 */
            unsigned long ticks;} time;         /* ...                        */
     unsigned char timtxt[15];                  /* decoded ascii time text    */
     for (i=0;i<2;i++)                          /* read 2 times               */
        {wchs(CR);                              /* new line                   */
         stype(gmetxt[i]);                      /* print header               */
         for (j=0;j<4;j++)                      /* read 4 time bytes          */
            {for (k=0;k<2;k++)                  /* read 2 ascii bytes         */
                if ((timtxt[k]=rdsg())==ABORT)  /* read time byte             */
                    {p_error(BADSAPMD);         /* strange response           */
                     return(0);};               /* ---> return                */
             timtxt[2]='\0';                    /* terminate string           */
             time.byt[j]=bhex(timtxt);};        /* convert hex ascii to int   */
         dcdtime(jtime,time);                   /* convert to ascii           */
         printf("%3d/%2d:%2d:%2d.%1d",jtime[0],jtime[1],jtime[2],jtime[3],
                jtime[4]);}}                    /* print time                 */
                                                /*                            */
/*************************************************                            */
/*                                                                            */
/*                              D C D T I M E                                 */
/*                                                                            */
/*      Convert binary time to ascii string.                                  */
/*                                                                            */
/*************************************************                            */
                                                /*                            */
dcdtime(btime,atime)                            /* decode time                */
    int btime[];                                /* binary time                */
    unsigned long atime;                        /* destination string         */
    {static unsigned long cnv[]={0x2a300l,7200l,120l,2l}; /* conversion const*/
     int i;                                     /* iteration variable         */
     for (i=0;i<4;i++)                          /* convert time               */
        {btime[i]=atime/cnv[i];                 /* get day                    */
         atime-=btime[i]*cnv[i];};              /* get remainder              */
12:                                             /* error exit                   */
                break;                          /* next                         */
                                                /*                              */
/*************************************************                              */
/*                                                                              */
/*      EOL                                                                     */
/*                                                                              */
/*************************************************                              */
                                                /*                              */
             case EOL:                          /* EOL                          */
                break;                          /* ignore blank line            */
                                                /*                              */
/*************************************************                              */
/*                                                                              */
/*      QUIT                                                                    */
/*                                                                              */
/*************************************************                              */
                                                /*                              */
             case Q:                            /* quit                         */
                scrup(0,0,24,79,0);             /* clear screen                 */
                i=inp(0x21);                    /* read 8259 interrupt mask     */
                outp(0x21,i|0x10);              /* stop serial interrupts       */
                exit(0);                        /* stop.                        */
             default:                           /* otherwise                    */
                error(BADCMD);}}}               /* unrecognized command         */
                                                /*                              */
/*************************************************                              */
/*                                                                              */
/*                              H E X W                                         */
/*                                                                              */
/*      Print the passed word in hex.                                           */
/*                                                                              */
/*************************************************                              */
                                                /*                              */
hexw(id,x)                                      /* display hex word             */
    struct window *id;                          /* window id                    */
    int x;                                      /* data                         */
    {hex(id,x>>8);                              /* display high ...             */
     hex(id,x);}                                /* ... and low                  */
                                                /*                              */
/*************************************************                              */
/*                                                                              */
/*                              H E X C                                         */
/*                                                                              */
/*      Convert the passed nibble to hex ascii.                                 */
/*                                                                              */
/*************************************************                              */
                                                /*                              */
char hexc(x)                                    /* convert to hex ascii         */
    int x;                                      /* nibble                       */
    {x&=0xf;                                    /* get nibble                   */
     return((x<=9)?x+'0':x-10+'A');}            /* convert to ascii             */
                                                /*                              */
/*************************************************                              */
/*                                                                              */
/*                              H E X                                           */
/*                                                                              */
/*      Print the specified byte at the current cursor position on the          */
/* specified window.                                                            */
/*                                                                              */
/*************************************************                              */
                                                /*                              */
hex(id,x)                                       /* print byte in hex            */
    struct window *id;                          /* window id                    */
    int x;                                      /* data                         */
    {wchw(id,hexc(x>>4));                       /* print high nibble            */
```

```
     purge();                                   /* empty garbage in buffer   */
     if (sgch(cmd)) return(0);                  /* issue command byte        */
     for (i=0;i<plen;i++)                       /* issue parameter bytes     */
        {chex(*par++,hxcmd);                    /* convert parameter byte    */
         for (j=0;j<2;j++)                      /* send bytes                */
            if (sgch(hxcmd[j]))                 /* issue byte and get resp.  */
                return(0);};                    /* good response?            */
     if (sgch(CR)) return(0);                   /* good termination?         */
     return(1);}                                /* return good completion    */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              S G C H                                      */
/*                                                                           */
/*      Send character to SAPMD and get response byte.                       */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
sgch(cmdb)                                      /* send character to SAPMD   */
    int cmdb;                                   /* command character         */
    {int rch;                                   /* response byte             */
     wrsg(cmdb);                                /* send byte                 */
     return(versg(cmdb));}                      /* return response           */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              V E R S G                                    */
/*                                                                           */
/*      Verify response from SAPMD                                           */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
versg(cmdb)                                     /* verify response from SAPMD*/
    int cmdb;                                   /* command character         */
    {int rch;                                   /* response byte             */
     rch=rdsg();                                /* get response byte         */
     if (rch==cmdb) return(0);                  /* good response?            */
     while (rch!=ABORT)                         /* abort command             */
        {wrsg(ILNK);                            /* send abort command        */
         rch=rdsg();};                          /* get response              */
     return(1);}                                /* return response           */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              D U M P                                      */
/*                                                                           */
/*      Read and display dumped data.                                        */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
dump(addr,len)                                  /* dump memory               */
    int addr,                                   /* address                   */
        len;                                    /* byte count                */
    {int i,                                     /* iteration variable        */
         j,                                     /* iteration variable        */
         k;                                     /* byte counter              */
     for (i=len;i>0;i-=16)                      /* count bytes displayed     */
        {wchs(CR);                              /* new line                  */
         hexw(&screen,addr);                    /* display address           */
         wchs(':');                             /* separate address          */
         wchs(' ');                             /*                           */
         addr+=16;                              /* next address              */
         k=0;                                   /* printed byte counter      */
         for (j=i>16?16:i;j>0;j--)              /* count bytes on line       */
            {wchs(k++==8?'-':' ');              /*                           */
             if (!rpbyte()) return(0);}}};      /* read and print byte       */
     wchw(id,hexc(x));}                         /* print low nibble          */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              C H E X                                      */
/*                                                                           */
/*      Convert byte to ascii hex string.                                    */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
chex(byt,str)                                   /* convert to ascii hex      */
    unsigned char byt;                          /* binary data               */
    char *str;                                  /* string                    */
    {*str=hexc(byt>>4);                         /* convert high nibble       */
     str++;                                     /* next byte in string       */
     *str=hexc(byt);}                           /* ... and low               */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              B H E X                                      */
/*                                                                           */
/*      Convert ascii hex string to int.                                     */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
bhex(hstr)                                      /* hex ascii string to int   */
    char *hstr;                                 /* character string          */
    {int a;                                     /* accumulator               */
     a=0;                                       /* clear accumulator         */
     while (*hstr!='\0')                        /* convert until end         */
        {a=a*16+(*hstr<='9'?*hstr-'0':*hstr-'A'+10); /* accumulate           */
         hstr++;};                              /* next character            */
     return(a);}                                /* return value              */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              E N T E R                                    */
/*                                                                           */
/*      Enter data into SC-1 memory.                                         */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
enter(cmd,cme,adr,maxadr,alen,dtype)            /* change target memory      */
    int cmd,                                    /* dump command              */
        cme,                                    /* enter command             */
        adr,                                    /* start address             */
        maxadr,                                 /* highest address           */
        alen,                                   /* address length            */
        dtype;                                  /* response type             */
    {int i,                                     /* iteration variable        */
         c,                                     /* character counter         */
         cflag;                                 /* change flag               */
     union {char bpar[3];                       /* parameter list            */
            int wpar;} cadr;                    /*                           */
     while (1)                                  /* forever ...               */
        {wchs(CR);                              /* new line                  */
         hexw(&screen,adr);                     /* display address           */
         wchs(':');                             /* colon ...                 */
         wchs(' ');                             /* ... and space             */
         for (i=0;i<8;i++)                      /* 8 bytes on line           */
            {cadr.wpar=adr++;                   /* plant address             */
             cadr.bpar[alen]=1;                 /* ... and byte count        */
             if (!sacmd(cmd,&cadr,cmd!=DUMPSFR?alen+1:1)||versg(dtype))
                {p_error(BADSAPMD);             /* display message           */
                 return(0);};                   /* ---> return               */
             if (!rpbyte()) return(0);          /* get contents              */
             wchs('.');                         /* terminate data             */
             wchs(' ');                         /* space                      */
             acc=0;                             /* clear accumulator          */
             c=2;                               /* count characters to go     */
             cflag=0;                           /* flag no change             */
             while (c>=0)                       /* read forever               */
                if ((ch=toupper(rdch()))!=(char)0xff)   /* character avail.   */
                    switch (ch)                 /* check for activation char */
                        {case '0':              /* hex digits ...             */
                         case '1':              /* ...                        */
                         case '2':              /* ...                        */
                         case '3':              /* ...                        */
                         case '4':              /* ...                        */
                         case '5':              /* ...                        */
                         case '6':              /* ...                        */
                         case '7':              /* ...                        */
                         case '8':              /* ...                        */
                         case '9':              /* ...                        */
                         case 'A':              /* ...                        */
                         case 'B':              /* ...                        */
                         case 'C':              /* ...                        */
                         case 'D':              /* ...                        */
                         case 'E':              /* ...                        */
                         case 'F':              /* ...                        */
                            if (c)              /* 2 characters yet?          */
                                {wchs(ch);      /* write character            */
                                 c--;           /* count 1 character          */
                                 acc=acc*16+(ch<='9'?ch-'0':ch-'A'+10);
                                 cflag++;};     /* flag byte changed          */
                            continue;           /* next                       */
                         case ' ':              /* space                      */
                            for (;c>-1;c--) wchs(' '); /* space to next col. */
                            if (cflag)          /* any change?                */
                                {cadr.bpar[alen]=acc; /* plant value          */
                                 if (!sacmd(cme,&cadr,alen+1) || rdsg()!=ACK)
                                    {p_error(BADSAPMD); /* display message    */
                                     return(0);}}; /* ---> return             */
                            break;              /* next                       */
                         case BACKSPACE:        /* oops                       */
                            cflag=0;            /* no change                  */
                         case CR:               /* carriage return            */
                            if (cflag)          /* any change?                */
                                {cadr.bpar[alen]=acc; /* plant value          */
                                 if (!sacmd(cme,&cadr,alen+1) || rdsg()!=ACK)
                                    {p_error(BADSAPMD); /* display message    */
                                     return(0);}}; /* ---> return             */
                            return(1);          /* return                     */
                         default:               /* invalid character          */
                            ;}}}}               /* ignore                     */
                                                /*                            */
/**************************************************                           */
/*                                                                            */
/*                              S A C M D                                     */
/*                                                                            */
/*      Issue command to SAPMD.                                               */
/*                                                                            */
/**************************************************                           */
                                                /*                            */
sacmd(cmd,par,plen)                             /* issue GSE command          */
    int cmd,                                    /* command byte               */
        plen;                                   /* number of parameter bytes */
    unsigned char *par;                         /* parameter bytes            */
    {unsigned char sch,                         /* response character         */
                   hxcmd[2];                    /* hex byte                   */
     int i,                                     /* iteration variable         */
         j;                                     /* iteration variable         */
     btime[4]=atime>0?5:0;}                     /* plant halfsec              */
                                                /*                            */
/*************************************************                           */
/*                                                                           */
/*                              P R O G                                      */
/*                                                                           */
/*      Program hex file into EEPROM.                                        */
/*                                                                           */
/*************************************************                           */
                                                /*                            */
prog()                                          /* program file               */
    {int i,                                     /* iteration variable         */
         cks,                                   /* checksum                   */
         bct,                                   /* byte count                 */
         off;                                   /* offset                     */
     union {char a[3];                          /* EEPROM address             */
            int b;} addr;                       /*                            */
     char *ip,                                  /* filename pointer           */
          *ipt;                                 /* ...                        */
     FILE *hxf;                                 /* hex file pointer           */
     char hxln[133];                            /* hex file record            */
     skbl();                                    /* skip to filename           */
     ip=iptr;                                   /* save filename start        */
     line[sizeof(line)-1]=CR;                   /* terminate command for sure*/
     off=0;                                     /* default offset             */
     while (*iptr!=CR && *iptr!=',') iptr++;    /* look for end-of-filename   */
     ipt=iptr;                                  /* mark end-of-filename       */
     if (scan()==COMMA)                         /* offset present?            */
        {schex();                               /* get hex address            */
         off=acc;                               /* change offset              */
         if (scan()!=EOL) return(BADCMD);};     /* good command?              */
     *ipt='\0';                                 /* terminate filename         */
     if ((hxf=fopen(ip,"rb"))==NULL) return(NOFILE); /* file exist?          */
     wchs(CR);                                  /* new line                   */
     stype("programming ...");                  /* acknowledge                */
     while (1)                                  /* loop forever               */
        {for (i=0;i<sizeof(hxln);i++)           /* read hex file record       */
            {hxln[i]=fgetc(hxf);                /* read byte                  */
             if (feof(hxf))                     /* early EOF?                 */
                {fclose(hxf);                   /* close file                 */
                 return(BADFILE);};             /* send message               */
             if (hxln[i]==LF) break;};          /* EOR?                       */
         if (hxln[0]!=':')                      /* good record?               */
            {fclose(hxf);                       /* early EOF?                 */
             return(BADFILE);};                 /* good record?               */
         switch (bhx(&hxln[7]))                 /* check type                 */
            {case 0:                            /* data record                */
                bct=bhx(&hxln[1]);              /* get byte count             */
                addr.a[1]=bhx(&hxln[3]);        /* get high address ...       */
                addr.a[0]=bhx(&hxln[5]);        /* ... and low                */
                ip=&hxln[9];                    /* point to data              */
                cks=bct+addr.a[1]+addr.a[0];    /* compute checksum           */
                addr.b+=off;                    /* adjust for offset          */
                for (i=0;i<bct;i++)             /* convert data to binary     */
                    {cks+=bhx(ip);              /* ... for checksum           */
                     ip+=2;};                   /* next byte                  */
                if ((-cks&0xff)!=bhx(ip))       /* checksums match?           */
                    {fclose(hxf);               /* close file                 */
                     return(BADFILE);};         /* return error               */
                ip=&hxln[9];                    /* point at data              */
                for (i=0;i<bct;i++)             /* program data               */
                    {addr.a[2]=bhx(ip);         /* plant data                 */
                     wchs(CR);                  /* new line ...               */
                     hexw(&screen,addr.b);      /* print address ...          */
                     wchs(':');                 /* separate data              */
                     wchs(' ');                 /*                            */
                     wchs(' ');                 /*                           */
                     hex(&screen,addr.a[2]);    /* print data                */
                     if (!sacmd(LOADEE,addr.a,3) || rdsg()!=ACK) /* issue cmd*/
                        {fclose(hxf);           /* close file                */
                         p_error(BADSAPMD);     /* signal error              */
                         return(0);};           /* return no error           */
                     ip+=2;                     /* next byte                 */
                     addr.b++;};                /* next address              */
                break;                          /* next record               */
            case 1:                             /* eof                       */
                return(0);                      /* file ok                   */
            default:                            /* else                      */
                return(BADFILE);}}}             /* strange file              */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              E X C F I L E                                */
/*                                                                           */
/*      Open command file.                                                   */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
excfile()                                       /* open command file         */
    {int i;                                     /* iteration variable        */
     skbl();                                    /* skip blanks to filename   */
     if (*iptr==CR) return(NOFILE);             /* null line?                */
     for (i=0;i<sizeof(line);i++)               /* stomp EOL                 */
        if (line[i]==CR) line[i]=0;             /* ...                       */
     if ((cfile=fopen(iptr,"rb"))==NULL)        /* file exist?               */
         return(NOFILE);                        /* return error              */
     else                                       /* file opened               */
        {cmdfile=1;                             /* flag command file open    */
         return(0);}}                           /* good file                 */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              B H X                                        */
/*                                                                           */
/*      Convert ascii hex byte to int.                                       */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
bhx(hstr)                                       /* hex ascii string to int   */
    char *hstr;                                 /* character string          */
    {int a,i;                                   /* accumulator               */
     a=0;                                       /* clear accumulator         */
     for (i=0;i<2;i++)                          /* convert until end         */
        {a=a*16+(*hstr<='9'?*hstr-'0':*hstr-'A'+10); /* accumulate           */
         hstr++;};                              /* next character            */
     return(a);}                                /* return value              */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              P E R R O R                                  */
/*                                                                           */
/*      Print error and purge SAPMD response buffer.                         */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
p_error(msg)                                    /* error and purge           */
    int msg;                                    /* message number            */
    {error(msg);                                /* signal error              */
     purge();}                                  /* empty response buffer     */
                                                /*                           */
                                                /*****************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                              S C A N                                      */
/*                                                                           */
/*      SCAN acquires user commands and turns them into lexical units for    */
/* processing by callers.  The integer returned is the token id, which is    */
/* also placed in the global variable 'token'.                               */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
#include <supglob.c>                            /* locate global data        */
                                                /*                           */
scan()                                          /*           SCAN            */
    {acc=0;                                     /* clear number accumulator  */
     skbl();                                    /* skip blanks               */
     if (*iptr>='0' && *iptr<='9')              /* check for number          */
        {while (*iptr>='0' && *iptr<='9')       /* accumulate #              */
            acc=acc*10+*iptr++-'0';             /* add digit                 */
         return(token=NUMBER);}                 /* return a number           */
     else                                       /* check for identifier/mark */
        return(fid());}                         /* search for id/mark        */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              S K B L                                      */
/*                                                                           */
/*      Skip blanks.                                                         */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
skbl()                                          /* eat blanks                */
    {while (*iptr==' ') iptr++;}                /* skip blanks               */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              F I D                                        */
/*                                                                           */
/*      Match the longest string in the input line.                          */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
fid()                                           /* find identifier           */
    {int i;                                     /* iteration variable        */
     struct name {char *ntxt;                   /* identifer                 */
                  int tkn;};                    /* token                     */
     char *nm,                                  /* pointer to command text   */
          *npt;                                 /* input pointer             */
     static struct name idnt[]={"DE",DE,        /* dump code                 */
                                "DR",DR,        /* dump ram                  */
                                "DS",DS,        /* dump SFR                  */
                                "EE",EE,        /* enter EEPROM              */
                                "ER",ER,        /* enter ram                 */
                                "ES",ES,        /* enter SFR                 */
                                "LA",LA,        /* rubber launch             */
                                "MON",MON,      /* monitor                   */
                                "P",P,          /* program                   */
                                "SG",SG,        /* set GMT                   */
                                "SM",SM,        /* set MET                   */
                                "TM",TM,        /* time                      */
                                "Q",Q,          /* quit                      */
                                ",",COMMA,      /* comma                     */
                                ":",COLON,      /* colon                     */
                                "/",SLASH,      /* slash                     */
                                "@",CMD,        /* at-sign                   */
                                "=",EQU};       /* equal                     */
     for (i=0;i<sizeof(idnt)/sizeof(struct name);i++) /* search for match    */
```

```
        {nm=idnt[i].ntxt;                       /* point at command          */
         for (npt=iptr;*nm==*iptr;iptr++) nm++; /* compare strings           */
         if (*nm=='\0')                         /* a match?                  */
            return(token=idnt[i].tkn);          /* ... yep, return token     */
         iptr=npt;};                            /* back up input             */
     if (*iptr==CR) return(token=EOL);          /* carriage return?          */
     return(token=BADCHAR);}                    /* strange identifier        */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              S C H E X                                    */
/*                                                                           */
/*      SCHEX attempts to read hex numbers.                                  */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
schex()                                         /* scan hex number           */
    {acc=0;                                     /* clear accumulator         */
     while (*iptr==' ') iptr++;                 /* skip blanks               */
     token=NUL;                                 /* guess missing number      */
     while (1)                                  /* loop till end-of-number   */
        switch (*iptr)                          /* what is the first char?   */
            {case '0':                          /* check for numbers         */
             case '1':                          /*               .           */
             case '2':                          /*               .           */
             case '3':                          /*               .           */
             case '4':                          /*               .           */
             case '5':                          /*               .           */
             case '6':                          /*               .           */
             case '7':                          /*               .           */
             case '8':                          /*               .           */
             case '9':                          /*               .           */
                acc=acc*16+*iptr++-'0';         /* accumulate                */
                token=NUMBER;                   /* a number                  */
                break;                          /* next                      */
             case 'A':                          /*               .           */
             case 'B':                          /*               .           */
             case 'C':                          /*               .           */
             case 'D':                          /*               .           */
             case 'E':                          /*               .           */
             case 'F':                          /*               .           */
                acc=acc*16+*iptr++-'A'+10;      /* accumulate                */
                token=NUMBER;                   /* a number                  */
                break;                          /* next                      */
             default:                           /* end-of-number             */
                return(token);};}               /* return                    */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              P R O M P T                                  */
/*                                                                           */
/*      Prompt for user keyboard input.                                      */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
prompt(prmpt)                                   /* ask user                  */
    char *prmpt;                                /* prompt string             */
    {scrup(24,0,24,79,0);                       /* erase line                */
     movcurs(24,0);                             /* position cursor           */
     for (;*prmpt!='\0';prmpt++) wch(*prmpt);}; /* write prompt string       */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              R D L N                                      */
/*                                                                           */
/*      Read keyboard input until an activation character is encountered.    */
/*                                                                            */
/*************************************************                            */
                                                /*                            */
rdln()                                          /* read input                 */
    {int j;                                     /* iteration variable         */
     static char splch[]={'H','P','M','K',      /* special chars. (keypad)    */
                          'Q','I','R','S',      /*                            */
                          'G','O',';','<'};     /*                            */
     static char spltkn[]={UP,DOWN,RIGHT,LEFT,  /* special tokens (keypad)    */
                           PGDN,PGUP,INS,DEL,   /*                            */
                           HOME,ND,CTA,CTR};    /*                            */
     iptr=line;                                 /* point to buffer start      */
     while (1)                                  /* poll until return          */
         if ((ch=rdch())!=(char)0xff)           /* character available?       */
            switch (ch)                         /* check for activation char */
                {case CTRLC:                    /* check for control-C        */
                    j=inp(0x21);                /* get 8259 mask              */
                    outp(0x21,j|0x10);          /* stop serial interrupts     */
                    popcurs();                  /* restore any saved cursor   */
                    exit(0);                    /* exit                       */
                 case CR:                       /* carriage return            */
                    *iptr=CR;                   /* plant character            */
                    iptr=line;                  /* point at start of input    */
                    return(EOL);                /* return character           */
                 case SPL:                      /* special characters         */
                    ch=rdch();                  /* get 2nd char in sequence   */
                    for (j=0;j<sizeof(splch);j++) /* look for special char.   */
                        if (splch[j]==ch)       /* is this it?                */
                            {if (spltkn[j]==DEL) /* del?                      */
                                {ch=BACKSPACE;  /* fake backspace             */
                                 goto l1;};     /* handle as backspace        */
                             *iptr=CR;          /* plant character code       */
                             iptr=line;         /* point at start of input    */
                             return(spltkn[j]);}; /* return it                */
                    break;                      /* not found, ignore          */
l1:                                             /* for DEL                    */
                 case BACKSPACE:                /* backspace                  */
                    if (line!=iptr)             /* characters on line?        */
                        {wrch(ch);              /* backspace ...              */
                         wrch(' ');             /* ... stomp character ...    */
                         wrch(ch);              /* ... and backspace again    */
                         --iptr;};              /* back up index              */
                 case LF:                       /* ignore line feed           */
                    break;                      /* next character             */
                 default:                       /* other characters           */
                    if (echo)                   /* keep characters?           */
                        {*iptr++=toupper(ch);   /* plant character            */
                         wrch(ch);}}}           /* echo it                    */
                                                /*                            */
/*************************************************                            */
/*                                                                            */
/*                              R D C H                                       */
/*                                                                            */
/*      Read keyboard input until a character is encountered. Poll for        */
/* SAPMD status.                                                              */
/*                                                                            */
/*************************************************                            */
                                                /*                            */
rdch()                                          /* read character             */
    {unsigned char ch;                          /* returned character         */
     if (!polst())                              /* any status data?           */
        {pshcurs();                             /* ... yep, save cursor pos. */
         status();                              /* display status             */
         popcurs();};                           /* restore cursor position    */
     if (cmdfile)                               /* command file open?         */
        {ch=fgetc(cfile);                       /* get high nibble ...        */
```

```
         if (!(ch==CEOF) && (!feof(cfile))) return(ch); /* EOF?               */
         fclose(cfile);                         /* close file                 */
         cmdfile=0;};                           /* mark command file closed   */
     return(rdc());}                            /* read keyboard              */
                                                /*                            */
/**************************************************                           */
/*                                                                            */
/*                              R D S G                                       */
/*                                                                            */
/*      Read SAPMD input until a character is detected. Poll for              */
/* SAPMD status.                                                              */
/*                                                                            */
/**************************************************                           */
                                                /*                            */
rdsg()                                          /* read SAPMD                 */
    {while (polsg())                            /* any response from SAPMD?   */
        if (!polst())                           /* check for status data      */
            {pshcurs();                         /* ... yep, save cursor pos. */
             status();                          /* display status             */
             popcurs();};                       /* restore cursor position    */
     return(rds());}                            /* read SAPMD character       */
                                                /*                            */
                                                /*****************************/
```

```
/******************************************************************************/
/*                                                                            */
/*                              D E B U G                                     */
/*                                                                            */
/*      Display 1st menu and process options.                                 */
/*                                                                            */
/**************************************************                           */
                                                /*                            */
#include <supglob.c>                            /* locate global data         */
                                                /*                            */
debug(argc,argv)                                /* debug SGA                  */
    int argc;                                   /* argument count (main)      */
    char *argv[];                               /* argument list (main)       */
    {int i,                                     /* temp.                      */
         j;                                     /* iteration variable         */
     static char *statxt[]=                     /* status line text           */
     {"EEPROM-ON    SELF-TEST    GSE    ACQUISITION    COMPLETE    ERROR",0};
     comio(0,0xc3);                             /* 9600; 8 bits; 1 stop; no p*/
     for (i=0;i<sizeof(sapmd)/sizeof(struct cal);i++) sapmd[i].serial=-1;
     if ((cfile=fopen("sapmd.cal","rt"))!=NULL) /* calibration file present? */
        {for (j=0;j<sizeof(sapmd)/sizeof(struct cal);j++)/* SAPMD cal. coefs.*/
            if ((fscanf(cfile,"%d%d%f%\n",&sapmd[j].serial,&sapmd[j].offset,
                 &sapmd[j].coef))!=3)
                 break;                         /* bail out                   */
         j=feof(cfile);                         /* check for eof              */
         fclose(cfile);};                       /* close cal file             */
     show(m4);                                  /* display status window      */
     sat(0x0f);                                 /* set obg                    */
     pmenu(m4,statxt,0,0);                      /* show status line           */
     sat(0x07);                                 /* restore normal video       */
     while (1)                                  /* loop forever ...           */
        {menu(1,1);                             /* display menu               */
         if (!j)                                /* check for cal file error   */
            {error(BADCAL);                     /* flag error                 */
             j++;};                             /* remove error               */
         while (1)                              /* not quite forever ...      */
            {prompt("SELECT OPTION: ");         /* prompt for user input      */
             rdln();                            /* get input, act. char.      */
             if (scan()==NUMBER)                /* check for option           */
                {i=acc;                         /* save option                */
                 if (scan()==EOL)               /* check for number only      */
                    {switch (i)                 /* number, process option     */
                        {case 1:                /* COMMAND/INTERROGATE        */
                            cmdint();           /* go poke SAPMD              */
                            break;              /* next                       */
                         case 2:                /* SAPMD SELF-TEST            */
                            selftest();         /* exercise SAPMD             */
                            break;              /* next                       */
                         case 3:                /* RECOVER PRESSURE DATA      */
                            recover();          /* go dump EEPROM log         */
                            continue;           /* next                       */
                         case 4:                /* DISPLAY PRESSURE DATA      */
                            if (dipress())      /* display data               */
                                {error(NOFILE); /* display error              */
                                 continue;};    /* don't redraw               */
                            break;              /* next                       */
                         case 5:                /* PRINT PRESSURE DATA        */
                            if (prpress()) error (NOFILE); /* print data      */
                            continue;           /* next                       */
                         default:               /* bad option                 */
                            error(BADCMD);      /* send message               */
                            continue;};         /* next iteration             */
                     break;}};                  /* next iteration             */
             if (token==Q)                      /* quit?                      */
                {scrup(0,0,24,79,0);            /* clear screen               */
                 i=inp(0x21);                   /* read 8259 mask             */
     outp(0x21,i|0x10);               /* stop serial interrupts    */
     exit(0);};                       /* stop.                     */
 if (token==CMD)                      /* command file?             */
    {if (i=excfile()) error(i);}      /* open command file         */
 else                                 /* not a command file?       */
    if (token!=EOL) error(BADCMD);}}} /* null line?                */
                                      /*                           */
                                      /****************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                              D I P R E S S                                */
/*                                                                           */
/*      Display pressure data file.                                          */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
#include <supglob.c>                            /* locate global data        */
                                                /*                           */
dipress()                                       /* display data              */
    {int i,                                     /* iteration variable        */
         pnum;                                  /* page number               */
     if (!getfile()) return(1);                 /* get file name             */
     screen.lines++;                            /* add line for bottom       */
     pshcurs();                                 /* save cursor attributes    */
     menu(5,1);                                 /* display window            */
     prpage(prsam);                             /* display 1st page          */
     pnum=0;                                    /* current data index        */
     echo=0;                                    /* clear echo flag           */
     while (1)                                  /* loop forever              */
        switch (rdln())                         /* read keypad               */
            {case PGUP:                         /* page up                   */
             case UP:                           /*                           */
                if (pnum>0)                     /* not 1st page?             */
                    {pnum-=40;                  /* back up page              */
                     prpage(&prsam[pnum]);};    /* display data              */
                break;                          /* next                      */
             case PGDN:                         /* page down                 */
             case DOWN:                         /*                           */
                if (&prsam[pnum+40]<samptr)     /* more data?                */
                    {pnum+=40;                  /* page down                 */
                     prpage(&prsam[pnum]);};    /* display data              */
                break;                          /* next                      */
             case HOME:                         /* bail out                  */
             case LEFT:                         /*                           */
                screen.lines--;                 /* restore screen size       */
                echo++;                         /* echo characters again     */
                popcurs();                      /* restore cursor            */
                return(0);                      /* ---> return               */
             default:                           /* else                      */
                ;}}                             /* ignore                    */
                                                /*                           */
/*************************************************                           */
/*                                                                           */
/*                              P R P A G E                                  */
/*                                                                           */
/*      Display a page of pressure data.                                     */
/*                                                                           */
/*************************************************                           */
                                                /*                           */
prpage(sm)                                      /* display page              */
    struct sam huge *sm;                        /* starting sample           */
    {static char *headr=                        /* window header             */
     {"SAMPLE               PRESSURE           SAMPLE               PRESSURE"};
     int i,                                     /* iteration variable        */
         j,                                     /* iteration variable        */
         k;                                     /* iteration variable        */
     clear(m6);                                 /* erase window              */
     m6->cury=1;                                /* first line                */
     m6->curx=6;                                /* space                     */
     wtype(m6,headr);                           /* print heading             */
     for (i=8;i<=48;i+=40)                      /* 2 columns                 */
        {m6->cury=2;                            /* move to top of window     */
         m6->curx=32;                           /* ... for serial #          */
         cursor(m6);                            /* ...                       */
         printf("SAPMD SERIAL #%5d",sm->serial); /* print serial #           */
         for (j=0;j<20;j++)                     /* 1 column                 */
            if (sm<samptr)                      /* good sample?             */
                {m6->curx=i;                    /* position cursor          */
                 m6->cury++;                    /* ...                      */
                 cursor(m6);                    /* ...                      */
                 printf("%3d                  %5.2f",sm->sample, sm->press);
                 sm++;}}                        /* next sample              */
     movcurs(24,1);                             /* hide cursor              */
     setcurs(0x3000);}                          /* one line long            */
                                                /*                          */
/**************************************************                         */
/*                                                                          */
/*                              G E T F I L E                               */
/*                                                                          */
/*       Get pressure data file and process.                                */
/*                                                                          */
/**************************************************                         */
                                                /*                          */
getfile()                                       /* display data             */
    {FILE *pfile;                               /* pressure data file       */
     unsigned char pdata[8192],                 /* pressure data            */
                   *pdptr;                      /* pressure data pointer    */
     union {unsigned char byt[4];               /* time bytes               */
            unsigned long tm;} tick;            /*                          */
     char ah[3];                                /* hex ascii character buffer*/
     union {int i;                              /* sample index             */
            unsigned char b[2];} s,             /*                          */
                                serial;         /* SAPMD serial number      */
     int i,                                     /* iteration variable       */
         j,                                     /* iteration variable       */
         k,                                     /* iteration variable       */
         p,                                     /* eod index in pdata       */
         m,                                     /* # samples in file        */
         adjust,                                /* Bennie's constant        */
         pnum;                                  /* page number              */
     float coef;                                /* calibration coef.        */
     coef=.058594;                              /* default FF=15.0 psi      */
     adjust=0;                                  /* default offset           */
     prompt("ENTER FILENAME: ");                /* prompt for file          */
     if ((i=rdln())==HOME || i==LEFT) return(0); /* read filename           */
     skbl();                                    /* skip blanks to filename  */
     if (*iptr==CR) return(0);                  /* null line?               */
     for (i=0;i<sizeof(line);i++)               /* stomp EOL                */
        if (line[i]==CR) line[i]=0;             /* ...                      */
     if ((pfile=fopen(iptr,"rb"))==NULL)        /* file exist?              */
         return(0);                             /* return error             */
     else                                       /* file opened              */
        {for (p=0;;p++)                         /* eof?                     */
             {ah[0]=fgetc(pfile);               /* get high nibble ...      */
              ah[1]=fgetc(pfile);               /* ... and low              */
              ah[2]='\0';                       /* terminate string         */
              if (feof(pfile)) break;           /* EOF?                     */
              pdata[p]=bhex(ah);};              /* convert to binary        */
         fclose(pfile);                         /* close file               */
         pdptr=pdata;                           /* point to pressure data   */
         samptr=prsam;                          /* point to processed data  */
             serial.b[1]=*pdptr++;              /* get serial number        */
             serial.b[0]=*pdptr++;              /* ...                      */
             for (j=0;j<sizeof(sapmd)/sizeof(struct cal);j++)/* cal. coef.  */
                if (sapmd[j].serial==serial.i)  /* is this one calibrated?  */
                    {coef=sapmd[j].coef;        /* set cal. coef.           */
                     adjust=sapmd[j].offset;    /* set calibration offset   */
                     break;};                   /*                          */
             for (j=0;j<4;j++) tick.byt[j]=*pdptr++; /* get GMT             */
             tick.tm*=51;                       /* convert to 100 msec units */
    m = ( p - 6 ) / 2;                 /* find amount of data       */
    for (k=0;k<m;k++)                  /* scan pressure data file   */
       {s.i=k;                         /* establish index           */
        s.b[0]=*pdptr++;               /* get sample number         */
        samptr->sample=s.i;            /* number sample             */
        j=*pdptr++;                    /* convert from char. to int */
        samptr->press=(j<adjust?0:j-adjust)*coef;  /*               */
        samptr->serial=serial.i;       /* mark source SAPMD         */
        samptr++;}};                   /* next sample               */
return(1);}                            /* good file                 */
                                       /*                           */
                                       /*****************************/
```