

Technical Document **2966**
June 1997

**Air Vehicle Diagnostic
System: CH-46 Aft Main
Transmission Fault
Diagnosis—Final Report**

K. G. Church
R. R. Kolesar
M. E. Phillips
R. C. Garrido

Approved for public release; distribution is unlimited.

19970912 024

DTIC QUALITY INSPECTED 8



Naval Command, Control and Ocean Surveillance Center
RDT&E Division, San Diego, CA 92152-5001

Technical Document **2966**
June 1997

**Air Vehicle Diagnostic
System: CH-46 Aft Main
Transmission Fault
Diagnosis—Final Report**

K. G. Church
R. R. Kolesar
M. E. Phillips
R. C. Garrido



DTIC QUALITY INSPECTED 3

Naval Command, Control and Ocean Surveillance Center
RDT&E Division, San Diego, CA 92152-5001

**NAVAL COMMAND, CONTROL AND
OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5001**

H. A. WILLIAMS, CAPT, USN
Commanding Officer

R. C. KOLB
Executive Director

ADMINISTRATIVE INFORMATION

The work detailed in this report was performed for the Chief of Naval Operations by the Naval Command, Control and Ocean Surveillance Center RDT&E Division, Advanced Technology Development Branch, Code D374.

Released by
R. R. Kolesar, Head
Advanced Technology
Development Branch

Under authority of
D. W. Murphy, Head
Advanced Systems
Division

CONTENTS

1 INTRODUCTION	1
2 DATA.....	3
2.1 CH-46 Helicopter	3
2.2 TEST STAND DATA	3
2.3 INFLIGHT DATA COLLECTION	4
2.4 FLIGHT TRIALS DATA.....	4
3 SYSTEM DESCRIPTION.....	7
3.1 NEURAL NETWORK PROCESSOR (N3P) SYSTEM DESIGN	7
3.1.1 System Summary	7
3.1.2 Software Processing Flow	7
3.2 NEURAL NETWORK PROCESSOR HARDWARE.....	9
3.2.1 Digital Storage Devices	9
3.2.2 Neural Network Processor Cards	9
3.3 NEURAL NETWORK PROCESSOR/HUMS INTERFACE	10
3.3.1 General Operation.....	10
3.3.2 Regime Processor	11
3.3.3 Regime Processor to N3P Message Content	12
3.3.4 N3P to Regime Processor Message Content	12
3.3.5 Regime Processor to MDAU.....	12
4 NEURAL NETWORK PROCESSOR LABORATORY BENCH TESTS	17
4.1 INTRODUCTION	17
4.1.1 Probability of False Alarm.....	17
4.1.2 Correction Condition Identification (CCI)	17
4.1.3 Correct Classification.....	18
4.2 DIGITAL PERFORMANCE TEST.....	18
4.2.1 Test Phase I - Baseline Digital Performance Test	18
4.2.2 Test Phase II - Digitally Degraded Sensor Environment Performance Test	19
4.2.3 Test Phase III - Digital DAFC Transition False Alarm Performance Test.....	22
4.3 DIGITAL PERFORMANCE TEST RESULTS	22
5 DETECTION OF A FAULT IN BU NO. 156432	29
5.1 USE OF IN-FLIGHT DATA	29
5.2 RADIAL BASIS FUNCTION NEURAL NETWORKS.....	29
5.3 AUGMENTING THE TRAINING OF THE RBF NETWORK USING IN-FLIGHT DATA ..	33
5.4 INDICATION OF AN ANOMALOUS CONDITION DURING CROSS VALIDATION	34
5.5 INVESTIGATION OF ANOMALY.....	35
5.6 NOTIFICATION OF NAVAL AVIATION SYSTEMS COMMAND (NAVAIR) REGARDING THE ANOMALY	36
5.7 RESULTS OF THE ENGINEERING INSPECTION (EI)	37
5.8 AUTO-ASSOCIATOR NEURAL NETWORK.....	38

6 NEURAL NETWORK PROCESSOR (N3P) FLIGHT TRIALS	43
6.1 INTRODUCTION	43
6.2 PROCESSING THE RESULTS OF THE TRIAL	45
6.3 PROCESSED FLIGHT TRIALS DATA	49
6.4 FLIGHT TRIALS CONCLUSIONS	50
7.0 PROGRAM CONCLUSIONS	51
7.1 RECOMMENDATIONS	51
APPENDIX A	A-1

Figures

1 N3P System in the VME chassis	7
2 AVDS block diagram	8
3 Regime Processor serial link signal flow diagram	11
4 Regime Processor to N3P message content	13
5 N3P to Regime Processor message content	14
6 Regime Processor to MDAU message content	15
7 Typical architecture of a RBF neural network. This network has a single hidden layer of RBFs fully interconnected to an output layer of linear units. Each output of an RBF node is multiplied by an interconnection weight that is determined by training. The weighted outputs of the RBF nodes are then summed at the corresponding linear unit. After summation, the output of the linear node is determined using a sigmoid transform to bound the output	30
8 Modified RBF in two-dimensional feature space formed using the modified logistic function $\left[1 + \exp\left(\frac{ x - \mu }{\sigma^2} - \phi\right)\right]^{-1}$, where ϕ represents the bias added to the receptive field, σ represents the width of the field, and μ represents the center of the field. Note that the function gives rise to a symmetric, localized, receptive field	31
9 Sigmoid transfer function that takes the sum of inputs to a linear output neuron and bounds the output response to [0,1]. Note that the input can vary from $-\infty$ to ∞ in two dimensions, but the output will always be "squashed" to between zero and one. Hence the nickname, squashing function	26

10	Typical Whisker plots showing the distribution of RBF Output Node Response: (a) typical no-defect data; (b) data from Bu No. 156432 at 27% torque; (c) data from Bu No. 156432 at 30% torque. The bars indicate the standard deviation of the distributions while the boxes contain 75% of the data with the mean of the distribution indicated by the horizontal bars within the box. For the typical no-defect response, only the first class of no-defect node is highly excited, with all other nodes showing a low, approximately constant, level of excitation. For the data from Bu No. 156432, all node responses are relatively low level, with the node having the highest response toggling as a function of torque.....	34
11	Trend in the SSE as a function of training epoch. The SSEs have been normalized by the maximum sum-squared error encountered during the training run. The SSE for the training data set is plotted as a solid line. The SSE for the three A/C in the cross-validation data set that did not have an anomalous response are shown as the lightly shaded lines while Bu No. 156432 is shown as the darkly shaded line	37
12	Damage to spiral bevel ring gear (a) general and (b) close-up. Damage occurs around the entire gear and across a significant portion of the mesh. Galling has worn through the black metal oxide coating and is close to cold metal working	39
13	Damage to spiral bevel input pinion gear (a) general and (b) close-up. The damage is symmetric to that of the mating spiral bevel ring gear. Again, the galling has worn through the black metal oxide coating and is close to cold metal working	39
14	Damage to starboard helical input pinion gear. Gouge or chip is localized to a single tooth near the edge of the mesh.....	39
15	Architecture of a typical AA network. This network has three layers that are fully interconnected. The hidden or middle layer is of a reduced number of nodes when compared to the input or output layer in order to encode the essence of the training class of data. The input and output layer have the same number of nodes. After training, the output of a feature vector is compared to the input and the RMS error is calculated between the two. The error is then compared to a threshold. If the error is greater than the threshold, the input is considered novel.....	40
16	Blind testing RMS prediction error vs. feature vector run for the trained AA network. The feature vectors from A/C Bu No. 156432 are in the upper portion of the graph while the feature vectors from the three other A/C in the cross-validation test set are in the graph's lower portion. The prediction error threshold is set at approximately 0.75.....	41
17	Front view of the VMEbus installation in AC 692. the VMEbus chassis containing the N3P is located in the lower portion of the equipment rack. The OQAR that logged the N3P status message is the rectangular black box shown in the upper left of the equipment rack.....	44

18 Side view of the VMEbus installation	45
19 Sample histogram from Flight N3SHA-6 showing distribution report indicators broken down into fields of good transmission, anomalous-faulty transmission, no report at this time, and network confidence failures. During this flight, multiple sensor/wiring failures related to loose connectors and an accelerometer backing off its mounting bolt caused network confidence failures	45
20 Sample histogram from Flight N3SHA-6 showing confidence indicated in the report indicators (figure 19). In this case, the majority of the no-fault answers provided by the RBF networks had marginal confidence while all answers provided by the Auto-Associator and some of the answers by the RBF networks had a relatively low confidence.....	48

Tables

1 Transmission/mix box test stand data summary.....	4
2 Test stand torques.....	5
3 Test Stand torques and corresponding flight conditions.....	5
4 N3P power budget.....	10
5 Acceptable torque levels for valid Data Acquisition Flight Conditions	11
6 Phase I digital baseline performance test.....	20
7 Phase II digitally degraded sensor environment test	21
8 Phase III digital DAFC transition false alarm test.....	24
9 Baseline digital performance test results by data record	25
10 Digitally degraded sensor environment performance test by data record	26
11 Digital DAFC transition false alarm performance test by data frame.....	27
12 Report indicator breakdown	46
13 DAFC breakdown.....	47
14 Report indicators broken down by flight. Flight N3SHA6-8 was particularly plagued with multiple sensor failures including bad microdot connections, accelerometers backing off mounts, problematic patch panel connections, intermittent signal saturation, etc.	49

1. INTRODUCTION

The goal of the helicopter drive system condition-monitoring component of the Air Vehicle Diagnostic System (AVDS) program is to develop technology that will facilitate transition to "condition-based maintenance" (CBM) for these components. The specific system requirements as stated in the AVDS Execution Plan are as follows:

- Detect and classify gearbox faults in real time in-flight.
- Cope with variations between gearboxes.
- Perform diagnostics in a variety of flight regimes.
- Reduce false-alarm rates.
- Reduce maintenance costs and facilitate transition to CBM.

This report describes the performance of a system developed by NRaD that meets these requirements.

2 DATA

2.1 CH-46 HELICOPTER

The focus of this project, because of its age and cost of operation, was on the CH-46 helicopter. The power train consists of two turbines that drive the forward and aft transmissions and subsequent rotors. Engine outputs are combined in a mix box. The power train is a constant- speed, variable-torque system. The effective torque varies as a function of the load on the rotor blades. Operationally, the aft-main transmission and mix box assembly have been the most troublesome. Because of these difficulties, this assembly has also been the subject of significant false removals, that is, premature aircraft removal and rebuild (NADEP Cherry Point estimates 30% of rebuilds are on aft-main transmission/mix box assemblies that are defect-free).

Instrumentation consisted of eight accelerometers and an optical tachometer. The accelerometers are uni-axial and attached to modified case nuts symmetrically positioned about the center-line of the assembly. The optical tachometer was substituted for the rotor position indicator. The tachometer produces 256 pulses per shaft revolution. The rotor position indicator shaft rotates at a nominal 114.4 Hz that dictates a data sampling rate of at least 58.5 kHz.

2.2 TEST STAND DATA

The data set consists of vibration data recorded from the aft transmission and mix box installed on a laboratory test stand. The data collection utilized a single transmission and mix box. Data were collected for two defect-free reassemblies of the mechanical system. Additionally, eight different faulted components were installed one at a time. Only one faulted component was present in the assembly for any given test. Data were recorded at nine torque levels on an International Recording Instruments Group B (IRIG-B) 28-channel tape recorder. The data were then transcribed into the digital domain using high-order, linear-phase anti-aliasing filters and a multi-channel, simultaneous sampling/digitization system. The following faulted components were installed and vibration data collected:

- Planetary Gear (Corrosion Spalling)*
- Spiral Bevel Input Pinion Bearing (Corrosion)*
- Spiral Bevel Input Pinion Gear (Tooth Spalling)*
- Helical Input Pinion Gear (Chipping)*
- Clutch Bearing (Brinelling)*
- Helical Idler Gear (Crack Propagation)
- Collector Gear (Crack Propagation)
- Quill Shaft (Crack Propagation)

* Component fault naturally occurring. Two components at different defect levels utilized. Components provided by NADEP Cherry Point.

Table 1 is a comprehensive list of transmission/mix box data that are known to have at least one 20-second, repeatably locatable (via time-code), contiguous block during which all nine channels are good (eight accelerometer and one tachometer). The data blocks have been examined for quality assurance using both automated and manual techniques. In order to locate such data blocks it was necessary to scroll through the entire analog recording cycle that corresponded to a valid torque level and for which ground-truth was known, using overlapping data blocks. In cases where no data were indicated, an error-free block of 20 seconds could not be found in the original analog data record.

Table 1. Transmission/mix box test stand data summary.

	27%T	30%T	40%T	45%T	50%T	55%T	60%T	70%T	75%T	80%T	100%T	FAULT LEVEL
No Defect, First Test										X		N/A
Planetary Pinion Bearing Corrosion											X	2
S.B. Input Pinion Bearing Corrosion	X		X	X	X		X	X	X	X	X	1, 2
S.B. Input Pinion Gear Scuffing	X		X	X	X		X	X	X	X	X	1, 2
M.B. Input Pinion Gear Chipping								X	X	X	X	2
M.B. Idler Gear Crack								X	X	X	X	N/A
Collector Gear Crack	X		X		X		X	X	X	X	X	N/A
Quill Shaft Crack	X		X	X	X		X	X	X	X	X	N/A
S.B. Assembly Galling	X	X				X	X	X	X	X	X	N/A
No Defect, Second Test	X		X	X	X		X	X	X	X	X	N/A

X — 20 seconds of good vibration data

2.3 IN-FLIGHT DATA COLLECTION

Vibration data were collected from nine deployed Marine Corps CH-46E aircraft. The eight accelerometers, locations, modified case nut mounts, and tachometer were the same as those used in the test stand data collection. A 14-channel IRIG-B tape recorder was used to acquire the data. To observe the effects of naturally occurring wear on the transmission/mix box, two data collections were made on each aircraft at widely separated intervals (approximately 6 to 10 months apart). Since the aircraft were deployed, the assumption was that the data represented systems in good working condition with no defects.

To expand the utility of the test stand data to in-flight conditions, it was necessary to acquire the in-flight data in flight conditions that approximated those of the test stand. For these data collections, pilots flew the aircraft at a constant torque and tried to maintain straight and level flight. The achieved flight conditions, corresponding to the nine test stand torques shown in table 2. Table 3 lists in-flight transmission/mix box data that are known to have at least one 20-second, contiguous block during which all nine data channels are functioning.

2.4 FLIGHT TRIALS DATA

Flight trials of the NRaD-developed, real-time, in-flight data collection and diagnostic system were conducted on a single CH-46E aircraft at NAWC Patuxent River, MD. The accelerometers, locations, modified case nut mounts, and tachometer were the same as previously used. For the diagnostic system

Table 2. Test stand torques.

Torque Level (%)	Flight Condition
27	Below light on wheel
30	750 ft/min descent
40	80 knots air speed (KNAS)
45	60 KNAS
50	light on wheels
60	100 KNAS
70	Hover, in-ground effect
75	Hover, out-of-ground effect
80	120 KNAS
100	140 KNAS VNE

Table 3. Phase II Transmission and mix box in-flight data (all no fault).

	27%T	30%T	40%T	45%T	50%T	55%T	60%T	70%T	75%T	80%T	100%T
BuNo 156457 A											
B	X	X	X	X	X	X	X	X	X	X	X
BuNo 156420 A											
B	X	X	X	X	X	X	X				X
BuNo 157673 A											
B	X	X	X	X	X	X	X	X		X	X
BuNo 153395 A											
B	X	X		X	X	X	X	X	X	X	X
BuNo 153960 A											
B			X	X	X		X	X	X	X	X
BuNo 156432 A		X									
B	X	X			X	X	X	X	X	X	X
BuNo 154829 A	X		X	X		X					
B	X	X	X	X	X	X	X	X	X	X	X
BuNo 154815 A	X	X	X	X	X	X	X	X	X	X	X
B											

X — 20 seconds of good vibration data

to operate autonomously, an automatic flight regime recognition capability was required. This permitted the pilot to fly the aircraft without the constraints imposed during the in-flight data collections. The system would automatically acquire data when the aircraft was in a valid Data Acquisition Flight Condition (DAFC), previously defined in table 2. Teledyne Controls had the task of developing this capability and integrating it into the in-flight system. Teledyne also manufactured the Health and Usage Monitoring System (HUMS) that was evaluated during these flight trials.

The flight trials consisted of 18 dedicated flights and one non-dedicated flight. The data collection system operated during 17 of the 19 flights and collected a total of 22.37 hours of transmission/mix box vibration data.

3 SYSTEM DESCRIPTION

3.1 NEURAL NETWORK PROCESSOR (N3P) SYSTEM DESIGN

3.1.1 System Summary

The N3P system processes, in near real time, transmission/mix box data from CH-46 helicopters and evaluates the mechanical condition of the mechanical system. The in-flight N3P system consists of a VME chassis with the capability to digitize 10 analog data channels at a nominal sample rate of 117 kHz, process the data, and determine the transmission condition approximately every 3.1 seconds. The system also contains a data recording capability that saves all data, results, and communications to 8-mm digital tapes for post-flight analysis.

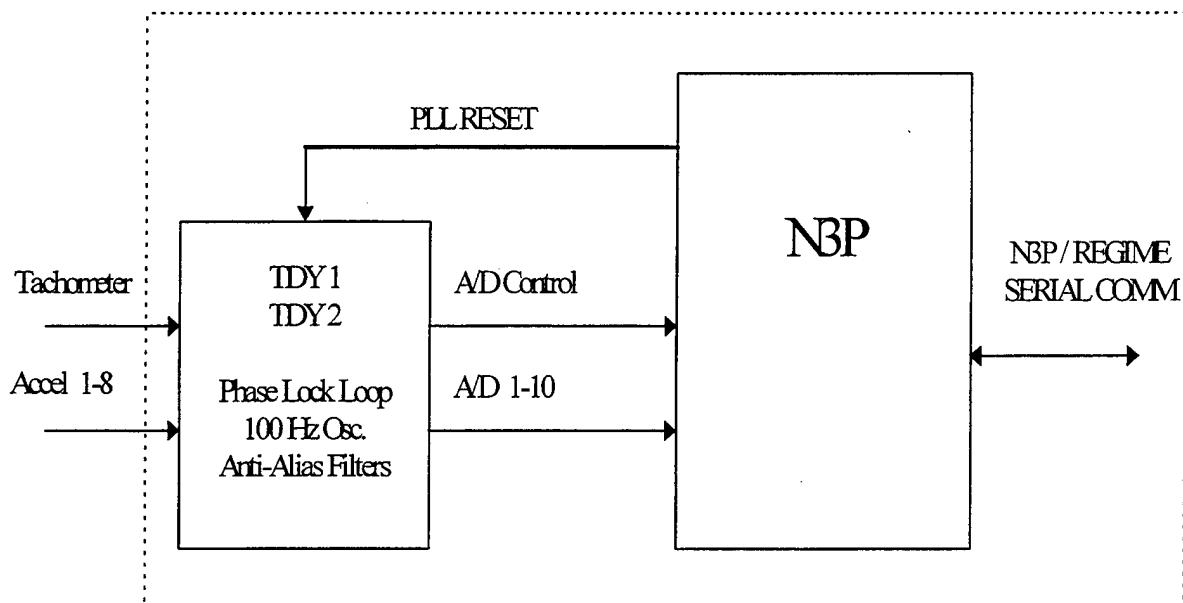


Figure 1. N3P System in the VME chassis.

3.1.2 Software Processing Flow

The N3P system has 10 processors that perform the various system functions. Analog data are received by the analog-to-digital (A/D) subsystem digitized synchronously with the transmission tachometer through a NRaD-designed Phase-Lock-Loop (PLL); see figure 1. The resultant digital time-series data are retrieved by the Data Distribution (DD) subsystem. The data are sent to the Quality Assurance (QA) and Feature Extraction (FE) subsystem. The Feature Vectors (outputs from the Feature Extraction) are sent to both a set of Radial Basis Function Neural Networks and a set of Auto-Associator Neural Networks, provided the time-series data pass the QA tests. Once the Neural Network outputs are generated, they are fused together. The fused RBF and AA results are then rated for acceptability. If acceptable they are averaged and an answer code is generated. The answer code is then assembled into the N3P output message, providing that valid Data Acquisition Flight Condition (DFAC) exists (see section 3.3). Figure 2 is the system block diagram.

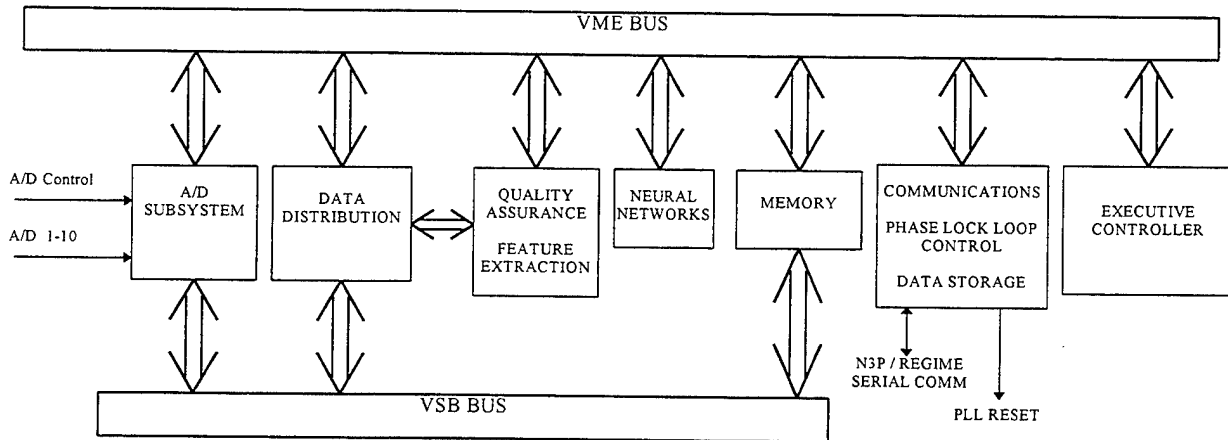


Figure 2. AVDS block diagram.

3.1.2.1 Executive Controller (HOST). The N3P process starts with the Executive Controller (HOST) execution. The Executive Controller does the bulk of the decision making on how the N3P processing is performed. The HOST also provides all the system services that the N3P requires to load and execute all of the program threads. After all the programs are loaded and executing, the HOST signals the DD subsystem to begin data distribution and the A/D subsystem to begin data acquisition. Once this is done, the HOST waits for results to arrive. The HOST only communicates over the VME subsystem bus to the other N3P processes (with the exception of the I/O Support Subsystem that can receive a reset over a serial interface and control information over the Ethernet).

When signaled that the QA/FE data products are available, the HOST checks the QA communication packet for errors. If no fatal errors are present, the HOST transfers the Feature Vector to the NN Subsystem for processing. The system design achieves fault tolerance through the "banks" of neural networks. If a single channel fault occurs, the HOST determines the transmission condition from the network within the bank that contains valid results. Otherwise, all network outputs are fused to determine the transmission state. The QA/FE results are always sent to the Memory Subsystem and bundled with the data frame sent by the DD.

When signaled that the NN processing is completed, the HOST retrieves the results from all the separate neural networks and fuses them into a single result word. The NN results are transferred to the Memory Subsystem. An answer code is also written to the Memory Status Section for transmission via the Communications process of the I/O Support subsystem.

3.1.2.2 Data Distribution (DD). After initialization, the DD Subsystem monitors the A/D Subsystem for the data ready signal. Data ready triggers the process to begin assembling data and demultiplexing it from the raw stream format onto 64K sample blocks. Once the data frame (64K samples $\times$ 10 channels) is completed, it is sent to the QA/FE Subsystem and the Memory Subsystem. The QA/FE Subsystem is notified that a new data frame is available and the DD starts assembly of the next data frame. Dual buffers (A/B) are employed throughout to increase system throughput.

3.1.2.3 Quality Assurance/Feature Extraction (QA/FE). The QA/FE Subsystem waits for the data ready signal from the DD then processes the data frame to determine that the data is of sufficient quality for NN processing (no dropouts, saturation, etc.) After quality checks, the feature set is

generated and the HOST is signaled that processing is complete and data products are available. The QA/FE Subsystem process then waits for another data frame ready and the processing repeats.

3.1.2.4 Neural Networks (NN). The NN Subsystem, after receiving the signal that a new Feature Vector has arrived, processes the features through both a Radial Basis Function neural network system and an Auto-Association neural network system. Each network system consists of a "bank" of eight neural networks that provide for fault tolerance. By eliminating a single channel from each of the individual networks, any one channel can be faulty and there is still a single network running that can determine the transmission state. After processing is complete, the HOST is signaled and the NN process waits for another Feature Vector to arrive.

3.1.2.5 I/O Support. The I/O Support Subsystem consists of three basic processes: the Communications process, the Phase Lock Loop Control process, and the Data Storage process, which provide the critical functions of external interface support and timing.

The Communication process receives and sends messages over a RS-232 interface. It also calculates and maintains the Data Acquisition Flight Condition (DFAC) Status and queues for N3P Answer processing. These processes supply the HOST generated answer under "valid" DFAC conditions and NO-ANSWER, otherwise.

The Phase Lock Loop (PLL) Control process responds to analog data acquisition and QA errors and can reset the external PLL hardware over a digital line. It also has the responsibility of maintaining the socket (Ethernet) interface with the HOST used for I/O Support Subsystem control.

The Data Storage process waits for the HOST to indicate that a complete data packet (data frame, QA/FE results and NN results) is ready and then records the data packet to a bank of three Exabyte 8505 8-mm digital data recorders.

3.2 NEURAL NETWORK PROCESSOR HARDWARE

The VME enclosure contains circuit cards that perform the following functions: 1) interface and signal conditioning for eight accelerometers and an optical tachometer; 2) flight regime recognition and RS-232 communications between HUMS and the N3P; and 3) the Neural Network Processor (N3P). Software, developed under this program, is implemented in the N3P hardware.

The in-flight VME enclosure is a ruggedized and electromagnetic interference (EMI)-shielded ELMA chassis. It has a 14-slot backplane and a six-slot VSB overlay. Table 4 shows a circuit card power budget.

3.2.1 DIGITAL STORAGE DEVICES

The chassis has a Kingston Technology 1GB hard disk. Three 8-mm Exabyte tape drives provide archival data storage of eight accelerometer and optical tachometer signals and eight intermediate system processing results.

3.2.2 NEURAL NETWORK PROCESSOR CARDS

The following 6U size VME circuit cards constitute the N3P system:

- (1) Force 2CE Single Board Computer

- (1) Motorola MVME 166 with MVME 7 12-006 Transition module
- (2) SKY bolt 8146-V
- (1) Micro Memory 256MB memory
- (1) ICS-110A Digitizer

Table 4. N3P power budget.

Device	+5 Volts	+12Volts	-12 Volts
1GB hard disk (EST).	2.55	2.50	
Three EXABYTE 8505XL tape drives	5.40	2.25	
Slot 1-FORCE 2CE	4.00	1.60	0.20
Slot 2-MVME 166	6.50	1.00	0.10
Slot 3-MVME712-006	0.28	0.10	0.15
Slot 4-MVME712-007	0.28	0.10	0.15
Slot 7-SKYbolt 8146-V	8.40		
Slot 8-SKYbolt8146-V	6.40		
Slot 10-256Mbyte Memory	7.75		
Slot 11-ICS-110A A/D	3.00	0.20	0.20
Slot 12-Spare			
Sum of currents	44.56	7.75	0.80
Watts	222.80	93.00	9.60
Total Watts	325.40		

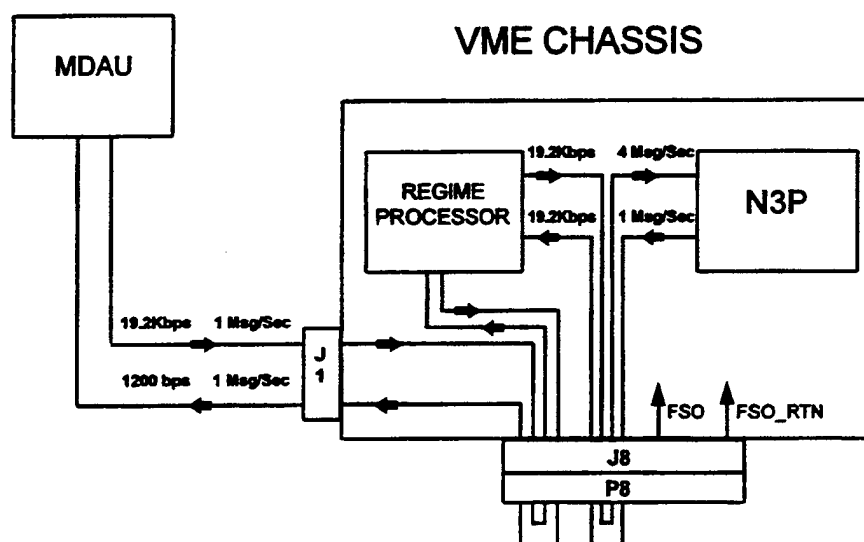
3.3 NEURAL NETWORK PROCESSOR/HUMS INTERFACE

3.3.1 General Operation

The HUMS MDAU communicates with the N3P via a Matrix I/O board in the VME chassis. The board provides three serial ports and processes aircraft flight data to determine valid Data Acquisition Flight Conditions (DAFC) for the N3P. A valid DAFC is achieved when the aircraft is in steady-state flight and is at one of the nine torque levels utilized in the transmission/mix box data collection phase of the project (discussed in section 2). Acceptable torque levels are shown in table 5. It is important to note that the only flight data available to the N3P are whether the aircraft is in a valid DAFC or not. The aircraft could be on the ground (27% torque) or at 140 knots (100% torque) and still be in a valid DAFC.

Table 5. Acceptable torque levels for valid Data Acquisition Flight Conditions (DAFC).

3.3.2 Regime Processor



The regime processor analyzes flight data associated with each 0.25-second interval and determines if the aircraft is in a valid DAFC for that interval. Determination of valid DAFCs is in accordance with a predefined set of flight parameters.

3.3.3 Regime Processor to N3P Message Content

Each 0.25 seconds the Regime Processor sends a message (figure 4) to the N3P. This message contains the current Regime Processor status and the most recently determined DAFC. The date and time values are derived from the MDAU internal time-of-day clock. The values sent to the N3P are adjusted for transit time, so that the N3P stores a consistent time with minimal processing. The values in each message field are formatted in hexadecimal.

3.3.4 N3P to Regime Processor Message Content

The N3P sends an event message to the regime processor once each second. This message (figure 5) contains the current status and the present condition of the transmission/mix box assembly. The values in each message field are hexadecimal.

3.3.5 Regime Processor to MDAU

The content of the four most-recent N3P event messages and the current regime are sent to the MDAU once each second for recording on the Optical Quick Access Recorder (OQAR). Figure 6 shows the message format.

BYTE#	CONTENT
1	Status
2	Spare
3	DAFC
4	Aircraft Type
5	Aircraft Tail ID
6	Message Year
7	Message Month
8	Message Day of Month
9	Message Hour
10	Message Minutes
11	Message Seconds
12	Checksum

Figure 4. Regime Processor to N3P message content.

BYTE #	CONTENT
1	Status
2	Spare
3-4	Fault
5	Message Hour
6	Message Minute
7	Message Second
8	Message Year
9	Message Month
10	Message Day of Month
11	Event Hour
12	Event Minute
13	Event Second
14	Checksum
11	Event Hour
12	Event Minute
13	Event Second
14	Checksum

Figure 5. N3P to Regime Processor message content.

BYTE 1								BYTE 2								BYTE 3								BYTE 4								BYTE 5								BYTE 6							
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
STATUS								FAULT								M.HOURS								M.MIN								M.SEC								M.YEAR							

BYTE 7								BYTE 8								BYTE 9								BYTE 10							
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
M.MON				M.DOM				E.HOUR				E.MIN				E.SEC				DAFC 1											

BYTE 11								BYTE 12								BYTE 13								BYTE 14								BYTE 15												
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8					
DAFC 2				DAFC 3				DAFC 4				FRAME COUNT								00				CKSUM																				

Legend:

STATUS	STATUS	08 bits	E. HOUR	EVENT HOUR	05 bits
FAULT	FAULT	16 bits	E. MIN	EVENT MINUTE	06 bits
M.HOURS	MESSAGE HOURS	05 bits	E. SEC	EVENT SECOND	06 bits
M. MIN	MESSAGE MINUTE	06 bits	DAFC 1	Fourth Most Recent Calculated DAFC	06 bits
M. SEC	MESSAGE SECOND	06 bits	DAFC 2	Second Most Recent Calculated DAFC	06 bits
M. YEAR	MESSAGE YEAR	07 bits	DAFC 3	Third Most Recent Calculated DAFC	06 bits
M. MON	MESSAGE MONTH	04 bits	DAFC 4	Most Recent Calculated DAFC	06 bits
M. DOM	MESSAGE DAY OF MONTH	05 bits	00	Padding for byte alignment	02 bits
			CKSUM	CHECKSUM	08 bits

Figure 6. Regime Processor to MDAU message content.

4 NEURAL NETWORK PROCESSOR LABORATORY BENCH TESTS

4.1 INTRODUCTION

The objectives of these tests was to quantify the system performance and determine if the requirements stated in the AVDS execution plan are being met (discussed in section 1). To quantify the system performance, the following metrics are defined:

1. Probability of False Alarm (PFA) - the number of times an alarm condition was issued when the underlying ground-truth indicated that a no defect condition existed.
2. Correct Condition Identification (CCI) - correct alarm condition (alarm/no alarm) based on the ground-truth of the data.
3. Correct Classification (CC) - probability of correct classification given that an alarm condition existed according to the ground-truth of the data.

4.1.1 Probability of False Alarm

The PFA metric quantifies the system's potential for false alarm. A high PFA indicates that the system has a tendency to false alarm. A low PFA indicates that insofar as the limitations of the testing procedures allow, the system appears to be robust and not subject to significant concerns regarding false-alarm rates.

4.1.2 Correction Condition Identification (CCI)

The CCI metric quantifies the system's performance in correctly issuing an alarm or no alarm condition.

4.1.2.1 CCI₁. A high CCI₁ indicates that the system performs well overall in correctly identifying the condition (alarm/no alarm) of the transmission. A low CCI₁ indicates that overall the system did not perform well in correctly identifying the condition of the transmission. Poor performance of the CCI₁ metric can be attributed to four factors: false positives (false alarm), false negatives, no reports (including those caused by poor data quality), and system failures.

4.1.2.2 CCI₂. A high CCI₂ indicates that when a condition could be determined for the transmission, the system performed well in correctly identifying this condition. A low CCI₂ indicates that specifically when condition reports were issued, the system did not perform well.

4.1.2.3 CCI₁ and CCI₂ Together. A low CCI₁ with a high CCI₂ indicates that the system performs well when a condition could be identified, but the system is plagued by no reports and/or system failures. Most likely, the no reports would be due to low neural network confidence, DAFC transition instabilities, and poor data quality.

A low CCI₁ with a low CCI₂ indicates that the system performed poorly in identifying the condition of the transmission mix box. This also indicates that the system made a substantial number of false positive or false negative determinations. If the PFA is high, the system made a significant number of false positive determinations. Otherwise, the system made a significant number of false negative determinations.

4.1.3 Correct Classification

The CC metric quantifies the system's performance in correctly classifying the transmission's condition to one of a set of predetermined states (currently eight defect states and one no-defect state).

4.1.3.1 CC₁. A high CC₁ indicates that the system performs well overall in correctly classifying the health of the transmission to a predetermined state. A low CC₁ indicates that overall the system did not perform well in correctly classifying the status of the transmission. Poor performance of the CC₁ metric can be attributed to five factors: misclassification, novel conditions that do not conform to one of the predetermined states, poor data quality, no reports, and system failures.

4.1.3.2 CC₂. A high CC₂ indicates that when a condition could be determined for the transmission, the system performed well in further correctly classifying this condition to one of a somewhat narrow set of defects or no defect. A low CC₂ indicates that specifically when condition reports were issued, the system did not perform well in further resolving the status of the transmission.

4.1.3.3 CC₁ and CC₂ Together. A low CC₁ with a high CC₂ indicates that the system classifies well when a condition could be identified, but the system is plagued by poor data quality, novel conditions, DAFC transition instabilities, or system failures.

A low CC₁ with a low CC₂ indicates that the system performed poorly in classifying the status of the transmission. This also indicates that the system made a substantial number of misclassifications. If the PFA is high, the system made a significant number of false alarm misclassifications. If the PFA is midrange to low, the system made a significant number of misclassifications across the set of classes.

4.2 DIGITAL PERFORMANCE TEST

Digital performance tests were carried out using test stand data only (section 2.2) stored on optical read-write media. The data records are each approximately 20 seconds long and consist of continuous digitally sampled data in, by definition, a valid Data Acquisition Flight Condition (DAFC). The N3P system requires 0.588 seconds of data to make a call. Therefore, 34 calls should be made over a 20-second period¹. These data have previously passed both automated and manual quality assurance checks and is not known to contain any sensor failures or artifacts.

4.2.1 Test Phase I - Baseline Digital Performance Test

4.2.1.1 Test Format. System performance for this phase of testing is qualified using the ground-truth of the data. The data records to be used for this phase of testing and the sequence of testing are listed in table 6. Note that the data records corresponding to no-fault data represent approximately 30% of the total data with the remaining 70% of data comprised of fault data. Also note that each subset of data is represented by a complete set of DAFCs (data availability permitting).

¹ Specifically, 34 calls would be made if the system had been up and running long enough to stabilize, the DAFC had been constant long enough to have been judged a stable DAFC and enough prior data had been accumulated to fill the integrator, etc.

4.2.1.2 Test Methodology. The purpose of Phase I Testing was to quantify the baseline digital performance of the N3P System. Phase I Testing will be based upon the correct issuance of an alarm or no alarm condition. An alarm condition is further broken down to reflect correct classification statistics. In addition, false alarm results are also be collected. For the purposes of Phase I Testing, each data frame, regardless of origin, has an equal weight in determining the overall statistics of the test. Each 20-second data record is played into the N3P System and the results will be tabulated in order to quantify the performance of the system.

4.2.1.3 Test Criteria. At the highest level, each data frame falls into one of two classes: (1) the class for which an alarm condition should be generated (defect) and (2) the class for which no alarm condition should be generated (no defect). Each data frame processed by the N3P System is graded to determine the performance of the system in determining the correct condition (alarm/no alarm) based on the ground-truth of the data. The output of this portion of Phase I Testing is the Correct Condition Identification (CCI) statistic. The CCI value is determined by dividing the number of data frames for which a correct answer was issued by the total number of data frames. This metric is calculated using two different sets of data. The first, CCI_1 , measures all data frames and all N3P outputs. The second, CCI_2 , is limited to those data frames for which the N3P system reported a condition. Consequently, a no condition report is considered an error by the CCI_1 metric but it is ignored by the CCI_2 metric. The difference between the two CCI values will quantify the effect of startup and transient on the system's stability. Each data record will be "cold-started²" into the N3P system to enhance repeatability.

In addition, the probability of correct classified (CC) and the probability of false alarm (PFA) are determined. The CC metric is the probability of correct classification given an existing alarm condition according to the ground-truth of the data. As before, the CC metric will be separated into two separate statistics. The first, CC_1 , measures all data frames and all N3P outputs while the second, CC_2 , measures only those frames for which the N3P system reported a condition. Again, the difference between the two CCs will quantify the effect of startup and transients on the system's stability.

The probability of false alarm is calculated using all of the no-defect data for which a condition report was issued. It is determined by dividing the number of data frames for which an alarm condition was issued when the underlying ground-truth indicated that no defect condition existed by the total number of data frames for which the N3P system issued a response. Note that this measure excludes those data frames for which no report was issued by the N3P System.

4.2.2 Test Phase II - Digitally Degraded Sensor Environment Performance Test

4.2.2.1 Data Format. System performance for this phase of testing is quantified using the ground truth of the data. In order to simulate a degraded sensor environment, a single channel of each record is digitally zeroed in software. This simulates the loss of signal associated with a channel. The data records to be used for this phase of testing, the sequence of testing, and the data channel to be zeroed for each record are listed in table 7. Note that the data records comprise the same basic subset of data used during Phase I Testing.

²i.e., the N3P System software is re-initialized between each data record.

Table 6. Phase I digital baseline performance test.

Test #	Tape #	Torque	Used	For	Defect Level	Defect	NRaD Number
2	5	100		Train	3	Planet Corrosion	02.B1.099.100
3	7	27	Test		2	SB Bearing Corrosion	03.A1.140.027
3	7	45	Test		2	SB Bearing Corrosion	03.A1.126.045
3	7	50		Train	2	SB Bearing Corrosion	03.A1.122.027
3	7	80		Train	2	SB Bearing Corrosion	03.A1.138.080
3	8	40	Test		2	SB Bearing Corrosion	03.A1.146.040
3	8	70	Test		2	SB Bearing Corrosion	03.B1.161.070
3	8	100	Test		2	SB Bearing Corrosion	03.A1.149.100
3	8	60		Train	2	SB Bearing Corrosion	03.B1.156.060
4	22	27	Test		4.5	SB Spalling	04.A1.426.027
4	22	40	Test		4.5	SB Spalling	04.A1.422.040
4	22	45	Test		4.5	SB Spalling	04.A1.421.045
4	22	50	Test		4.5	SB Spalling	04.A1.418.050
4	22	70	Test		4.5	SB Spalling	04.A1.410.070
4	22	80	Test		4.5	SB Spalling	04.A1.424.080
4	23	40	Test		4.5	SB Spalling	04.A2.432.040
4	23	45	Test		4.5	SB Spalling	04.A2.431.045
4	23	60	Test		4.5	SB Spalling	04.A1.433.060
5	15	70	Test		5	Port Input Pinion Chip	05.B1.294.070
5	15	75	Test		5	Port Input Pinion Chip	05.B1.295.075
5	15	100		Train	5	Port Input Pinion Chip	05.B1.291.100
5	16	80	Test		5	Port Input Pinion Chip	05.B1.300.080
6	21	80	Test		4	Port Idler Gear Crack	06.A1.404.080
6	21	100	Test		4	Port Idler Gear Crack	06.A1.405.100
6	24	70		Train	5	Port Idler Gear Crack	06.A1.440.070
6	24	80		Train	5	Port Idler Gear Crack	06.A1.445.080
7	80	100		Train	5	MB Collector Crack	07.A2.495.100
8	35	80	Test		0	None	00.A2.678.080
8	35	100	Test		0	None	00.A1.679.100
8	70	40	Test		10	Quill Crack	08.A1.280.040
8	70	50	Test		10	Quill Crack	08.A1.282.050
8	70	70	Test		10	Quill Crack	08.A1.284.070
8	70	80	Test		10	Quill Crack	08.A1.286.080
9	37	100	Test		0	None	00.A4.727.100

Table 7. Phase II digitally degraded sensor environment test.

Test #	Tape #	Torque	Used	For	Level	Defect	to Remove	NRaD Number
2	5	100		Train	3	Planet Corrosion	5	02.B1.099.100
3	7	27	Test		2	SB Bearing Corrosion	3	03.A1.140.027
3	7	45	Test		2	SB Bearing Corrosion	4	03.A1.126.045
3	7	50		Train	2	SB Bearing Corrosion	7	03.A1.122.027
3	7	80		Train	2	SB Bearing Corrosion	4	03.A1.138.080
3	8	40	Test		2	SB Bearing Corrosion	3	03.A1.146.040
3	8	70	Test		2	SB Bearing Corrosion	4	03.B1.161.070
3	8	100	Test		2	SB Bearing Corrosion	4	03.A1.149.100
3	8	60		Train	2	SB Bearing Corrosion	7	03.B1.156.060
4	22	27	Test		4.5	SB Spalling	7	04.A1.426.027
4	22	40	Test		4.5	SB Spalling	4	04.A1.422.040
4	22	45	Test		4.5	SB Spalling	4	04.A1.421.045
4	22	50	Test		4.5	SB Spalling	4	04.A1.418.050
4	22	70	Test		4.5	SB Spalling	7	04.A1.410.070
4	22	80	Test		4.5	SB Spalling	3	04.A1.424.080
4	23	40	Test		4.5	SB Spalling	7	04.A2.432.040
4	23	45	Test		4.5	SB Spalling	3	04.A2.431.045
4	23	60	Test		4.5	SB Spalling	3	04.A1.433.060
5	15	70	Test		5	Port Input Pinion Chip	4	05.B1.294.070
5	15	75	Test		5	Port Input Pinion Chip	1	05.B1.295.075
5	15	100		Train	5	Port Input Pinion Chip	2	05.B1.291.100
5	16	80	Test		5	Port Input Pinion Chip	1	05.B1.300.080
6	21	80	Test		4	Port Idler Gear Crack	4	06.A1.404.080
6	21	100	Test		4	Port Idler Gear Crack	4	06.A1.405.100
6	24	70		Train	5	Port Idler Gear Crack	2	06.A1.440.070
6	24	80		Train	5	Port Idler Gear Crack	2	06.A1.445.080
7	80	100		Train	5	MB Collector Crack	4	07.A2.495.100
8	35	80	Test		0	None	4	00.A2.678.080
8	35	100	Test		0	None	7	00.A1.679.100
8	70	40	Test		10	Quill Crack	3	08.A1.280.040
8	70	50	Test		10	Quill Crack	4	08.A1.282.050
8	70	70	Test		10	Quill Crack	7	08.A1.284.070
8	70	80	Test		10	Quill Crack	7	08.A1.286.080
9	37	100	Test		0	None	4	00.A4.727.100

4.2.2.2 Test Methodology. The purpose of Phase II Testing is to quantify the performance and robustness of the N3P System in the presence of a degraded signal environment. Phase II Testing will be based upon the correct issuance of an alarm or no-alarm condition. An alarm condition will be further broken down to reflect correct classification statistics. In addition, false alarm results are also collected. For the purpose of Phase II Testing, each data frame, regardless of origin, will have equal weight in determining the overall statistics of the test. Each 20-second data record is played into the N3P System and the results will be tabulated in order to quantify the performance of the system. As in Phase I, each data record will be cold-started into the N3P System.

4.2.2.3 Test Criteria. The same criteria used in Phase I of the testing is used to determine system performance during Phase II of the testing: CCI, PFA, and CC.

4.2.3 Test Phase III - Digital DAFC Transition False Alarm Performance Test

4.2.3.1 Data Format. System performance for this phase of testing is quantified using the ground-truth of the data. In order to simulate rapid transitions among DAFCs, a single data record of approximately 20 seconds duration is created. The data record will consist of data frames from randomly selected DAFCs of a single aircraft (A/C). The data frames are combined into a single data record of 34 frames to play into the N3P System. The data frames to be used for this phase of testing and the sequence of combination of the frames are listed in table 8.

4.2.3.2 Test Methodology. The purpose of Phase III Testing is to quantify the false alarm rate of the N3P System as a function of transition between DAFCs. Phase III Testing is based on upon the collection of false alarm statistics. For the purpose of Phase III Testing, each data frame, regardless of origin, will have equal weight in determining the false alarm rate. The data record is played into the N3P System and the results will be tabulated in order to quantify the performance of the system. The first frame of the data record is cold-started into the N3P System.

4.2.3.3 Test Criteria. The criteria used in Phase III of the testing to determine system performance is PFA.

4.3 DIGITAL PERFORMANCE TEST RESULTS

Phase I and II Combined					
Criteria	CCI	CC	PFA		
Value	99.9%	96.0%	0.0%		
Phase I - Baseline Digital Performance Test					
Criteria	CCI ₁	CCI ₂	CC ₁	CC ₂	PFA
Value	100.0%	100.0%	96.2%	96.2%	0.0%

Note: In all cases, the condition of the transmission according to ground-truth was correctly identified. Cases where the Correct Classification was not made occurred during the first and/or second data

frame of a record where the Radial Basis Function (RBF) nets had not yet stabilized. There were seven data frames (Helical Idler Gear Crack 80%T) where the AAs incorrectly identified the fault as normal. It is believed that this due to the threshold of the AAs being set to error on the side of no false alarm. The breakdown of calls for each data record used in Phase I Testing is included in table 9.

Phase II - Digitally Degraded Sensor Environment Performance Test

Criteria	CCI ₁	CCI ₂	CC ₁	CC ₂	PFA
Value	99.9%	99.9%	95.8%	95.8%	0.0%

Note: All cases where the correct condition of the transmission were not identified were false negatives (i.e., a ground-truth defect condition was identified by the anomaly detectors as a good transmission) that occurred at a DAFC of 70% Torque. Cases where the Correct Classification was not made are once again the first few data frames of a record where the RBF nets had not yet stabilized. There were 18 data frames (SB Bearing Corrosion, Helical Idler Gear Crack, and Quill Shaft Crack) where the AAs incorrectly identified the fault as normal. Once again, this believed to be due to the threshold of the AA's being set to err on the side of no false alarm. The breakdown of calls for each data record is included in table 10.

Phase III - Digital DAFC Transition False Alarm Performance Test

Criteria	PFA	CCI	CC
Value	0.0%	100.0%	97.0%

Note: There were no false alarms during this, or any, portion of testing. While the criteria of CCI and CC are not performance criteria for this test phase according to the test plan, these criteria were both satisfied by any sense of the measure. The breakdown of calls for each data record is included in table 11.

Table 8. Phase III digital DAFC transition false alarm test.

Flight #	Torque	Regime	Used	Defect		NRaD Number
				For	Level	
1614	40		Test		0	None 1614.040.14.38
1614	70	SF	Test		0	None 1614.070.15.34
1614	30	Descent	Test		0	None 1614.030.16.26
1614	27	BFPLOW	Test		0	None 1614.027.14.51
1614	60	SF	Test		0	None 1614.060.15.59
1614	60	SF	Test		0	None 1614.060.15.59
1614	100	VNE	Test		0	None 1614.100.15.44
1614	45		Test		0	None 1614.045.15.00
1614	50	SF	Test		0	None 1614.050.15.24
1614	80	SF	Test		0	None 1614.080.15.41
1614	27	BFPLOW	Test		0	None 1614.027.14.32
1614	27	BFPLOW	Test		0	None 1614.027.14.51
1614	50	LOW	Test		0	None 1614.050.15.03
1614	70	SF	Test		0	None 1614.070.15.56
1614	100	CLIMB	Test		0	None 1614.100.16.39
1614	45	SF	Test		0	None 1614.045.15.21
1614	27	BFPLOW	Test		0	None 1614.027.14.51
1614	45		Test		0	None 1614.045.14.40
1614	55	SF	Test		0	None 1614.055.15.29
1614	100	VNE	Test		0	None 1614.100.15.47
1614	80	SF	Test		0	None 1614.080.15.49
1614	50	SF	Test		0	None 1614.050.16.06
1614	27	DESCENT	Test		0	None 1614.027.16.31
1614	60	SF	Test		0	None 1614.060.15.31
1614	45	SF	Test		0	None 1614.045.16.09
1614	70	SF	Test		0	None 1614.070.15.56
1614	80	CLIMB	Test		0	None 1614.080.16.33
1614	75	SF	Test		0	None 1614.075.15.37
1614	30		Test		0	None 1614.030.14.35
1614	40	SF	Test		0	None 1614.040.15.19
1614	60	SF	Test		0	None 1614.060.15.59
1614	27	BFPLOW	Test		0	None 1614.027.14.51
1614	100	VNE	Test		0	None 1614.100.15.47

Note: If an NRaD number appears more than once, sequential data frames will be used from that data file. Otherwise, the first data file will be used.

Table 9. Baseline digital performance test results by data record.

#Calls Possible	1700					
#Calls Made	1700					
Defect	NRaD Number	CCI	CC	No Report	False Alarm	
Planet Corrosion	02.B1.099.100	34	33	0	0	
SB Bearing Corrosion	03.A1.140.027	34	32	0	0	
SB Bearing Corrosion	03.A1.126.045	34	33	0	0	
SB Bearing Corrosion	03.A1.122.027	34	33	0	0	
SB Bearing Corrosion	03.A1.138.080	34	33	0	0	
SB Bearing Corrosion	03.A1.146.040	34	33	0	0	
SB Bearing Corrosion	03.B1.161.070	34	33	0	0	
SB Bearing Corrosion	03.A1.149.100	34	33	0	0	
SB Bearing Corrosion	03.B1.156.060	34	33	0	0	
SB Spalling	04.A1.426.027	34	32	0	0	
SB Spalling	04.A1.422.040	34	33	0	0	
SB Spalling	04.A1.421.045	34	33	0	0	
SB Spalling	04.A1.418.050	34	33	0	0	
SB Spalling	04.A1.410.070	34	33	0	0	
SB Spalling	04.A1.424.080	34	33	0	0	
SB Spalling	04.A2.432.040	34	33	0	0	
SB Spalling	04.A2.431.045	34	33	0	0	
SB Spalling	04.A1.433.060	34	33	0	0	
Port Input Pinion Chip	05.B1.294.070	34	32	0	0	
Port Input Pinion Chip	05.B1.295.075	34	32	0	0	
Port Input Pinion Chip	05.B1.291.100	34	33	0	0	
Port Input Pinion Chip	05.B1.300.080	34	32	0	0	
Port Idler Gear Crack	06.A1.404.080	34	33	0	0	
Port Idler Gear Crack	06.A1.405.100	34	33	0	0	
Port Idler Gear Crack	06.A1.440.070	34	33	0	0	
Port Idler Gear Crack	06.A1.445.080	34	33	0	0	
MB Collector Crack	07.A2.495.100	34	33	0	0	
None	00.A2.678.080	34	33	0	0	
None	00.A1.679.100	34	32	0	0	
Quill Crack	08.A1.280.040	34	32	0	0	
Quill Crack	08.A1.282.050	34	32	0	0	
Quill Crack	08.A1.284.070	34	33	0	0	
Quill Crack	08.A1.286.080	34	32	0	0	
None	00.A4.727.100	34	32	0	0	
Defect						
None	1614.100.15.47	34	33	0	0	
None	3651.030.12.51	34	33	0	0	
None	3651.040.13.30	34	33	0	0	
None	3651.050.14.08	34	33	0	0	
None	3651.060.14.02	34	33	0	0	
None	3651.070.13.59	34	33	0	0	
None	3653.040.10.28	34	33	0	0	
None	3653.045.16.04	34	33	0	0	
None	3653.050.10.34	34	33	0	0	
None	7642.027.15.02	34	33	0	0	
None	7642.055.15.57	34	33	0	0	
FAULT	7645.030.14.01	34	31	0	0	
FAULT	7645.055.15.53	34	32	0	0	
FAULT	7645.070.15.47	34	32	0	0	
FAULT	7645.080.16.09	34	32	0	0	
None	7646.100.16.13	34	33	0	0	
		Totals	1700	1635	0	0

Table 10. Digitally degraded sensor environment performance test by data record.

#Calls Possible	1700					
#Calls Made	1700					
Proposed Accelerometer						
Defect	to Remove	NRaD Number	CCI	CC	No Report	False Alarm
Planet Corrosion	5	02.B1.099.100	34	32	0	0
SB Bearing Corrosion	3	03.A1.140.027	34	32	0	0
SB Bearing Corrosion	4	03.A1.126.045	34	33	0	0
SB Bearing Corrosion	7	03.A1.122.027	34	33	0	0
SB Bearing Corrosion	4	03.A1.138.080	34	33	0	0
SB Bearing Corrosion	3	03.A1.146.040	34	33	0	0
SB Bearing Corrosion	4	03.B1.161.070	34	33	0	0
SB Bearing Corrosion	4	03.A1.149.100	34	33	0	0
SB Bearing Corrosion	7	03.B1.156.060	34	33	0	0
SB Spalling	7	04.A1.426.027	34	31	0	0
SB Spalling	4	04.A1.422.040	34	33	0	0
SB Spalling	4	04.A1.421.045	34	33	0	0
SB Spalling	4	04.A1.418.050	34	33	0	0
SB Spalling	7	04.A1.410.070	34	33	0	0
SB Spalling	3	04.A1.424.080	34	33	0	0
SB Spalling	7	04.A2.432.040	34	33	0	0
SB Spalling	3	04.A2.431.045	34	33	0	0
SB Spalling	3	04.A1.433.060	34	33	0	0
Port Input Pinion Chip	4	05.B1.294.070	34	32	0	0
Port Input Pinion Chip	1	05.B1.295.075	34	32	0	0
Port Input Pinion Chip	2	05.B1.291.100	34	33	0	0
Port Input Pinion Chip	1	05.B1.300.080	34	32	0	0
Port Idler Gear Crack	4	06.A1.404.080	34	33	0	0
Port Idler Gear Crack	4	06.A1.405.100	34	33	0	0
Port Idler Gear Crack	2	06.A1.440.070	33	32	0	0
Port Idler Gear Crack	2	06.A1.445.080	34	32	0	0
MB Collector Crack	4	07.A2.495.100	34	33	0	0
None	4	00.A2.678.080	34	33	0	0
None	7	00.A1.679.100	34	31	0	0
Quill Crack	3	08.A1.280.040	34	33	0	0
Quill Crack	4	08.A1.282.050	34	32	0	0
Quill Crack	7	08.A1.284.070	34	32	0	0
Quill Crack	7	08.A1.286.080	34	32	0	0
None	4	00.A4.727.100	34	32	0	0
Defect						
None	1	1614.100.15.47	34	33	0	0
None	2	3651.030.12.51	34	33	0	0
None	3	3651.040.13.30	34	33	0	0
None	4	3651.050.14.08	34	33	0	0
None	6	3651.060.14.02	34	33	0	0
None	7	3651.070.13.59	34	33	0	0
None	4	3653.040.10.28	34	33	0	0
None	5	3653.045.16.04	34	33	0	0
None	6	3653.050.10.34	34	33	0	0
None	3	7642.027.15.02	34	33	0	0
None	5	7642.055.15.57	34	33	0	0
FAULT	7	7645.030.14.01	34	32	0	0
FAULT	4	7645.055.15.53	34	31	0	0
FAULT	7	7645.070.15.47	34	32	0	0
FAULT	4	7645.080.16.09	34	32	0	0
None	2	7646.100.16.13	34	32	0	0
Totals			1699	1629	0	0

Table 11. Digital DAFC transition false alarm performance test by data frame.

#Calls Possible	34				CCI	CC	No Report	False Alarm
#Calls Made	34				34	33	0	0
					Defect			
Flight #	Torque	Regime	Used	For	Level	Defect	NRaD Number	
1614	40		Test		0	None	1614.040.14.38	
1614	70	SF	Test		0	None	1614.070.15.34	
1614	30	Descent	Test		0	None	1614.030.16.26	
1614	27	BFPLOW	Test		0	None	1614.027.14.51	
1614	60	SF	Test		0	None	1614.060.15.59	
1614	60	SF	Test		0	None	1614.060.15.59	
1614	100	VNE	Test		0	None	1614.100.15.44	
1614	45		Test		0	None	1614.045.15.00	
1614	50	SF	Test		0	None	1614.050.15.24	
1614	80	SF	Test		0	None	1614.080.15.41	
1614	27	BFPLOW	Test		0	None	1614.027.14.32	
1614	27	BFPLOW	Test		0	None	1614.027.14.51	
1614	50	LOW	Test		0	None	1614.050.15.03	
1614	70	SF	Test		0	None	1614.070.15.56	
1614	100	CLIMB	Test		0	None	1614.100.16.39	
1614	45	SF	Test		0	None	1614.045.15.21	
1614	27	BFPLOW	Test		0	None	1614.027.14.51	
1614	45		Test		0	None	1614.045.14.40	
1614	55	SF	Test		0	None	1614.055.15.29	
1614	100	VNE	Test		0	None	1614.100.15.47	
1614	80	SF	Test		0	None	1614.080.15.49	
1614	50	SF	Test		0	None	1614.050.16.06	
1614	27	DESCENT	Test		0	None	1614.027.16.31	
1614	60	SF	Test		0	None	1614.060.15.31	
1614	45	SF	Test		0	None	1614.045.16.09	
1614	70	SF	Test		0	None	1614.070.15.56	
1614	80	CLIMB	Test		0	None	1614.080.16.33	
1614	75	SF	Test		0	None	1614.075.15.37	
1614	30		Test		0	None	1614.030.14.35	
1614	40	SF	Test		0	None	1614.040.15.19	
1614	60	SF	Test		0	None	1614.060.15.59	
1614	27	BFPLOW	Test		0	None	1614.027.14.51	
1614	100	VNE	Test		0	None	1614.100.15.47	
1614	30		Test		0	None	1614.030.14.54	

5 DETECTION OF A FAULT IN BU NO. 156432

5.1 USE OF IN-FLIGHT DATA

The transmission/mix box diagnostic neural networks were trained to recognize one of eight test stand conditions (seven different defect conditions and a no-defect condition). These training data were limited to a single mechanical assembly that underwent several rebuilds during the seeded fault trials. Additionally, the data were from an assembly that was installed in a massive test stand and driven by electric motors into a water brake load. To extend the diagnostics to a realistic environment, recorded in-flight vibration data were used. Sixty-four features indicative of the condition of a particular subassembly, in the transmission/mix box, were extracted from each of the eight vibration channels. These features were combined into a single feature of 512 elements that represented the overall condition of the transmission/mix box for a particular Data Acquisition Flight Condition. These exemplars were used to augment the original network training. This was expected to increase the robustness and reduce false alarms due to differences between the test stand and in-flight environments. The data, from multiple aircraft, could also be used to evaluate the capability to perform diagnostics on a transmission/mix box without first establishing a baseline signature. (The Teledyne HUMS required 25 in-flight hours to establish a baseline signature). Implicit in the use of in-flight data from deployed Marine Corps aircraft is the assumption that the data represent transmission/mix box assemblies in good working order with no defects.

5.2 RADIAL BASIS FUNCTION NEURAL NETWORKS

A neural network differs from classical techniques of mechanical diagnosis in that the network is trained using a representative sample of data out of the "universe" of data or features the network is expected to encounter. This training allows the network to form statistical models of the types of features that the network will be expected to recognize. Essentially, the network builds its own statistical defect models using examples from the feature sets that it is desired it recognize. Assuming that the training feature sets are truly representative of the particular classes of data, the statistical models are inherently robust and have an ability to generalize across the entire class of data.

This statistical model is then used to recognize other examples of the same type of defect. Since the models are robust, the network can generalize to features that are similar but not exactly the same as the training sets. It is desired that this robustness could be exploited so that the network would not have to be baselined using data from each and every transmission/mix box to which they would ever be applied. At this time, the program was using a Radial Basis Function (RBF) network as the major technique of classification since RBF networks have many characteristics that make them desirable to use in this application. Some of the characteristics that the program was trying to exploit are that RBFs work well as classifiers, that a RBF network forms "soft" as opposed to sharply delineated decision surfaces, and that RBF nets can be rapidly trained on a relatively large quantity of features.

RBF neural networks have the common network architecture of simple processing elements or nodes that are heavily interconnected and are typically constructed as a feed-forward network with a single hidden layer. Each node of the hidden layer simultaneously receives the entire input vector and is fully interconnected to an output layer of linear units (see figure 7). Each hidden layer node uses a radial basis function to form a localized receptive field that calculates the "closeness" of the input vector to a particular data cluster in feature space. The closer the input vector is to the center of the cluster the more strongly the neuron responds with an upper bound of one.

The RBF used is typically a gaussian basis or a logistic function. This type of function gives rise to a symmetric receptive field in feature vector space (figure 8). The combination of these basis functions in the hidden layer gives rise to "soft" decision surfaces where there is no hard boundary that encloses a decision region. This soft decision surface allows the RBF net's performance to degrade somewhat gracefully as a function of deterioration of data quality. As data quality degrades, the features extracted tend to fall further from the centers of the data clusters, resulting in a smaller output for any given class or combination of classes. This can be contrasted with the hard decision surfaces formed by a typical Multi-Layer Perceptron (MLP).

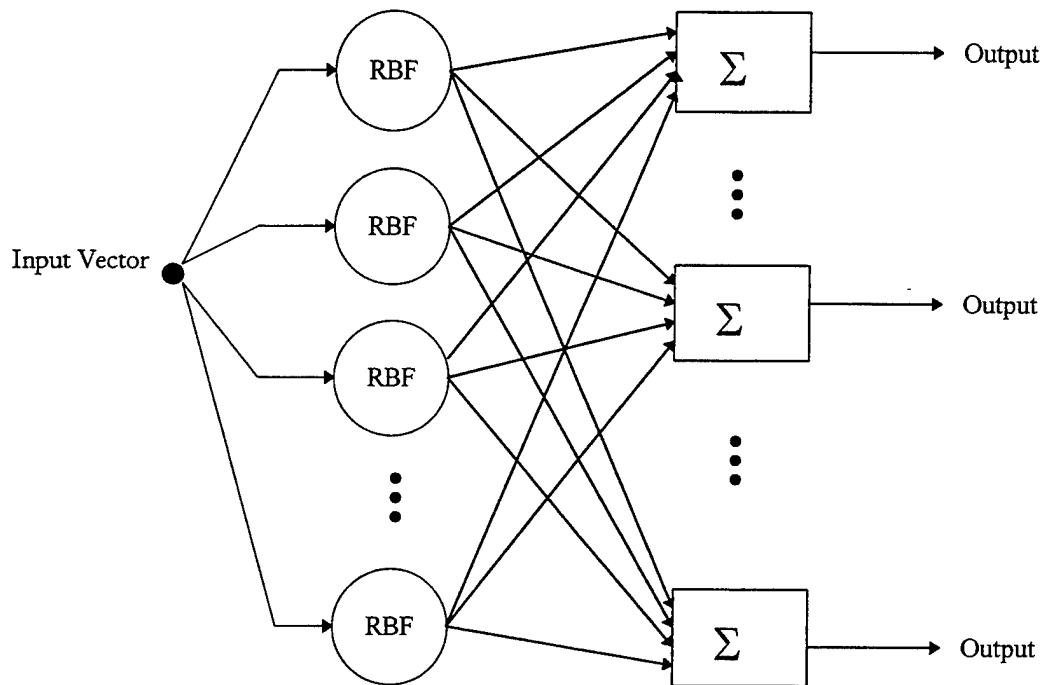


Figure 7. Typical architecture of a RBF neural network. This network has a single hidden layer of RBFs fully interconnected to an output layer of linear units. Each output of an RBF node is multiplied by an interconnection weight that is determined by training. The weighted outputs of the RBF nodes are then summed at the corresponding linear unit. After summation, the output of the linear node is determined using a sigmoid transform to bound the output.

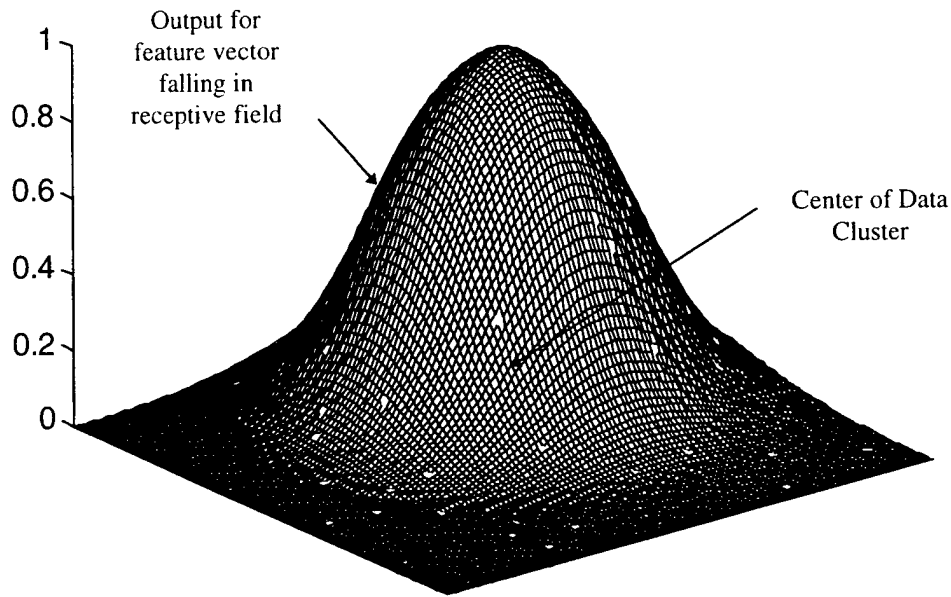


Figure 8. Modified RBF in two-dimensional feature space formed using the modified logistic function $\left[1 + \exp\left(\frac{|x - \mu|}{\sigma^2} - \phi\right)\right]^{-1}$, where ϕ represents the bias added to the receptive field, σ represents the width of the field, and μ represents the center of the field. Note that the function gives rise to a symmetric, localized, receptive field.

Each hidden unit of a multi-layer perceptron is used to form a hyperplane as opposed to a localized receptive field. This hyperplane, in combination with hyperplanes formed from other hidden layer nodes, encloses a decision region of feature space. Typically, it takes four or more hyperplanes to enclose a given decision region. The input feature vector is either inside or outside of this bounded decision region as opposed to the continuum of responses provided by a soft decision surface. This strictly delineated decision boundary is a potential source of high false positives for a multi-layer perceptron approach.

To form the RBF network, the features extracted from the data are first clustered in some manner to reduce the complexity of the problem and to speed computations. It is desired that this clustering be efficient without losing the essential character of the features. The best clustering is achieved when the clusters densely populate the data-rich regions of the feature space. This speeds training since most training techniques only update those weights associated with a stimulated cluster during any given training cycle and improves classification accuracy. Our approach, as is common in the neural network community, has been to use the k -means³ algorithm to cluster the feature training set. The choice of the number of clusters, k , is arbitrary and typically confirmed using cross-validation techniques. Due to the architecture of RBF networks, these types of nets perform very well on features that have well-separated data clusters.

³ J. MacQueen. 1967. "Some Methods for Classification and Analysis of Multivariate Observations," *In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume I, Statistics*. L. M. Le Cam and J. Neyman, eds. University of California Press, Berkeley, CA.

After the features have been clustered, a RBF corresponding to a single middle layer neuron is placed at the centroid of each cluster. By centering the RBF of this neuron on the given data cluster, the receptive field of this neuron has been tuned to respond to a feature vector input near the given data cluster. All of the RBF neurons are then connected in turn to all of the output class neurons with each output class represented by a single neuron in the output layer. This neuron is typically a linear unit that sums its interconnection weighted inputs, and then applies a sigmoid transform to bound the output between zero and one (figure 9). The network is then trained to recognize a particular class of data using a representative set of features corresponding to those data.

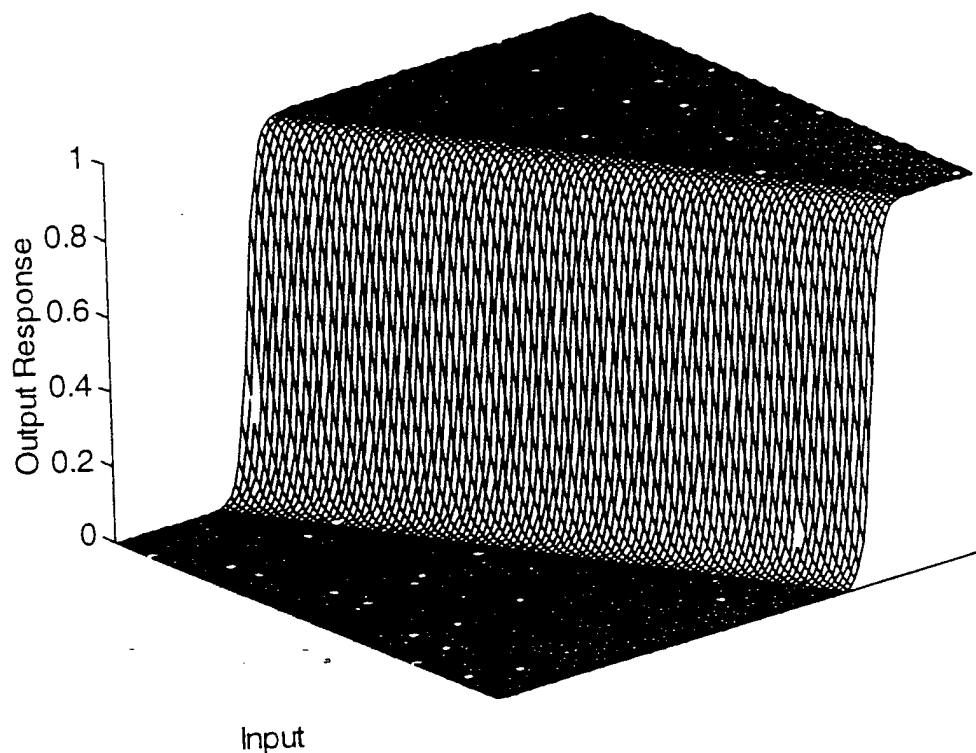


Figure 9. Sigmoid transfer function that takes the sum of inputs to a linear output neuron and bounds the output response to $[0, 1]$. Note that the input can vary from $-\infty$ to ∞ in two dimensions, but the output will always be "squashed" to between zero and one. Hence the nickname, squashing function.

Once the network architecture has been determined, the network can be trained and the interconnection weights determined. RBF networks are typically trained on one class of data at a time. During the training, representative features from the class being trained are presented to the network and the interconnection weights for the RBF nodes or data clusters are adjusted until the network does an adequate job recognizing this class of data. The next class of data are then trained with its associated weights adjusted and so forth, until all classes of data have been trained. The training is insensitive to the sequence of classes that is presented to the net. Once the network has been fully trained, it is ready to be cross-validated using hold-out techniques and other methods.

5.3 AUGMENTING THE TRAINING OF THE RBF NETWORK USING IN-FLIGHT DATA

In order to train and evaluate the performance of the RBF network using the in-flight data, a training feature set and a cross-validation set of features were required. Due to differences in the method of data collection between the in-flight and test-rig data sets, it was necessary to design a preprocessing filter transform to align the two data sets. This necessitated retraining the entire network since the baseline data had fundamentally changed. The same test-rig training set as used previously was reused along with a subset of the total in-flight data set.

During the training process, it was desired that the network develop a capability to generalize and to correctly classify as demonstrated using the cross-validation techniques. The need to develop these capabilities across a range of torques and airframes drove the design of the training set. Obviously, a representative sample of each defect was required with a corresponding sample of no-defect conditions. Since the no-defect condition is the nominal condition and was assumed to occur for a large number of transmissions, the amount of no-defect training data are over-represented relative to any given fault class but the amount of no-defect training data were roughly the same as the training data for all fault classes. It was also desirable for each training exemplar to be represented across the full range of torques available in order to provide for a robust classification capability. These concerns provided the framework for the partitioning of the in-flight data into a training set and a cross-validation set.

Given the total amount of in-flight data available, feature sets from four of the A/C were chosen to augment the training set and feature sets from the remaining four A/C were chosen for the evaluation set. This decision was based on the amount of data available from each A/C for both the initial and final data collections⁴, the torques available for all A/C, and the torques available for any given A/C. Note that no data from the A/C chosen for the cross validation feature set had ever been presented to the network during the training process. This type of "blind" cross validation using features from a distinctly new and separate, but similar type of data that the network should be able to correctly classify based on generalization capabilities, is commonly referred to as hold-out testing, if done properly. It is worth emphasizing that to be true hold-out testing, the data must be uniquely distinct from the data forming the training set and original to the network.

The training data set was then used to train the RBF network. First the features were clustered using a modified *k*-means algorithm. Several choices of the number of centroids were evaluated for the purposes of cross validation and network complexity tradeoff studies. The different networks were then trained on the same data set using supervised learning techniques. Each of the seven defect classes of training data from the test-rig were sequentially presented to each network and the interconnection weights adjusted until the network correctly classified the defect class with an arbitrarily low Sum-Squared Error (SSE). The combined no defect class of training consisting of in-flight and test-rig no-defect data were then presented to the networks in the same manner with corresponding adjustment to the interconnection weights. After training, no further adjustments were made to the networks and they were ready for cross validation.

⁴Due to instrumentation/sensor malfunctions, the initial data collections from five of the A/C were unusable and all of the data from one A/C proved unusable.

5.4 INDICATION OF AN ANOMALOUS CONDITION DURING CROSS VALIDATION

During the cross validation of the trained networks, data from three of the A/C were consistently and confidently classified as corresponding to a no-defect condition, as expected. The networks' response to the features extracted from the second data collection on Bu No. 156432, however, proved to be quite unexpected. For the two lowest torques at which data were collected, the networks consistently responded as if they were confused as to the condition of this particular transmission. In particular, the response for the network lying at the knee of the performance versus the complexity curve is shown in figure 10. Additionally, the classification at the higher torques was consistently no defect, but at a degraded confidence level. The response for all networks trained was consistent and similar.

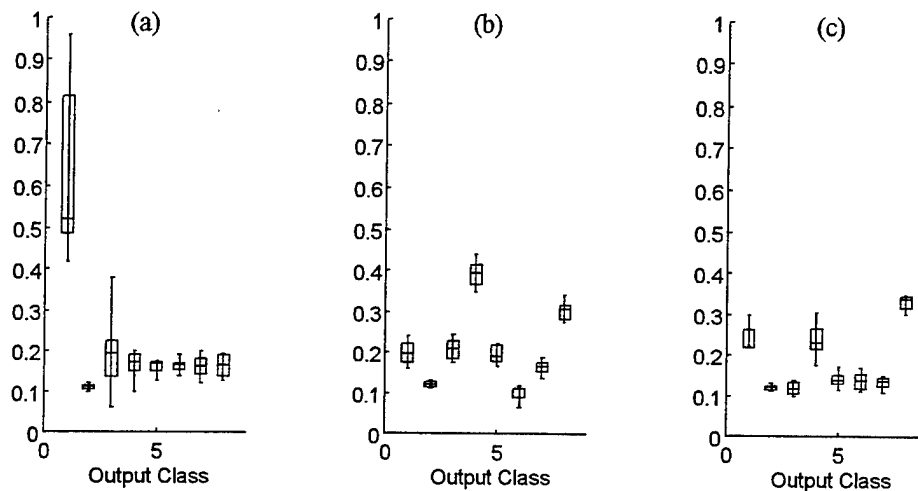


Figure 10. Typical Whisker plots showing the distribution of RBF Output Node responses: (a) typical no-defect data; (b) data from Bu No. 156432 at 27% torque; (c) data from Bu No. 156432 at 30% torque. The bars indicate the standard deviation of the distributions while the boxes contain 75% of the data with the mean of the distribution indicated by the horizontal bars within the box. For the typical no-defect response, only the first class or no-defect node is highly excited, with all other nodes showing a low, approximately constant, level of excitation. For the data from Bu No. 156432, all node responses are relatively low level, with the node having the highest response toggling as a function of torque.

In general, the RBF output nodes were showing a low-level response to all classes previously trained on, with no confident classification to any particular class of defect or no defect previously trained on. Further, the node that seemed to be most highly excited was not stable as a function of torque. For the two lowest torques, the highest node output toggled among classes four and eight. No significance is attached to the particular classes excited since the network was most likely responding to a type of data clustered outside of its "universe" and the scarcity of data makes analysis problematic.

This type of behavior from RBF classifiers is typically indicative of presentation to the network of a type of data with which they have no previous experience and have not been trained to recognize. If the data have not been corrupted in some manner (*i.e.*, "garbage in—garbage out") and most resembles the characteristics of a class of data clusters on which the network has been trained, the output class representing these clusters will be the one most highly excited. If the data do not resemble any of the characteristics of the data clusters, the response of a typical RBF net is not a priori determinable except in certain pathologic cases. The situation is analogous to training a RBF network to recognize red or green, and then presenting the network with a yellow and asking it to tell you whether it is red or green.

In some sense, it is kind of red, and kind of green, but the correct response is that it is neither and is novel to the network.

Given our past experience with these data, this type of behavior was atypical due to the nature of and consistency of the misclassification. Previously, we had found that there was a locally optimal number of data clusters for which the size of the RBF employed tended to be irrelevant (within reasonable limits) to the performance of the network. As the number of clusters increased, performance did not significantly increase and tended to depend on the size of the RBF employed. When the number of clusters reached an architecture and data-dependent threshold, adding more clusters decreased the performance independent of the size of the radial basis function. As the number of data clusters was decreased below the local optimum, performance was highly dependent on the size of the radial basis function used and tended to be degraded compared to the networks trained using the locally optimum number of clusters. In all cases when misclassifications were previously observed, they tended to be distributed among several to all of the data classes and inconsistent within any given class. Our experience with the features extracted from Bu No. 156432 was the first time that we had observed consistent misclassification confined to a single class for all of the network architectures used during the cross-validation procedure.

5.5 INVESTIGATION OF ANOMALY

Once the anomaly had been consistently indicated, the baseline data were once again examined regarding its integrity. The baseline automated quality assurance algorithms were rerun on the raw data with no deficiencies in the data indicated. These algorithms had been developed to check for sensor faults and data deficiencies that had previously been seen in the avionics environment. The algorithms also checked to ensure that the data collection system was functioning properly and did not induce any artifacts into the data.

The raw data were then manually examined in minute detail to determine if a new type of sensor fault or data collection, system-induced problem had been encountered during the collection of this particular data set. There was no indication from visual inspection of the raw data that data had been corrupted in any manner. To all intents and purposes, the raw time-series data appeared to be free from defects. The data did have slightly different amplitude levels on two of the eight channels (one elevated and one decreased) when compared to the minimal amount of data available from the initial collection⁵, but the magnitude of the changes were within normal deviations of all the data examined to date. The conclusion was drawn that the anomaly was not caused by corrupted data.

The training and cross-validation data sets were then repartitioned in order to further investigate the anomaly. Each of the three cross-validation set members that did not behave in an anomalous manner was sequentially switched with a single member of the initial training data set. Three RBF networks having a different number of k -means clusters and different cluster widths were then trained using the new training data set. The networks were then evaluated using the new cross-validation data set. This investigation continued until all original members of the training data set had been replaced in turn by one of the members of the original cross-validation set that did not display anomalous behavior. In total,

⁵ Due to instrumentation difficulties, only one complete data record of approximately 20 seconds duration was available from the first data collection. The anomaly may or may not have been present at this time but the short duration of the data record did not allow enhanced signal averages to be formed and very little analysis or comparison could be done with such a small amount of data.

27 different networks were trained⁶ and evaluated. In all cases, the behavior of the networks still indicated that the data from the low torques of Bu No. 156432 were different from the remainder of the in-flight data and the test-rig no-fault data, and that this difference was statistically significant.

Lastly, a generic feed-forward MLP network with two hidden layers was selectively trained using the baseline training data set and standard back propagation techniques of training. The back propagation algorithm was slightly modified so that during each training epoch, the cross-validation data were also presented to the network and a SSE-computed for the data from each of the A/C. Although the cross-validation data were presented during the training process, no adjustment of the interconnection weights was made for the cross-validation data. The errors due to the cross-validation data set were not allowed to back propagate through the net, only the errors due to learning to recognize the baseline training data set were allowed to back propagate and cause weight adjustments. As the network converged, the SSEs for all of the A/C from the cross-validation data set significantly decreased while the MSE calculated for the data from Bu No. 156432 stayed relatively large and roughly constant. This trend in the SSEs is shown in figure 11.

5.6 NOTIFICATION OF NAVAL AVIATION SYSTEMS COMMAND (NAVAIR) REGARDING THE ANOMALY

At this time, the decision was made to notify NAVAIR that an anomaly had occurred during the processing of the in-flight data and that the anomaly was indicative of a transmission that differed from the seven other transmissions available for comparison. Naval Air Warfare Center Aircraft Division (NAWCAD) PAX River Flight Test Engineering Group and NAVAIR 4.4.2 were contacted and informed that the data from A/C BuNo 156432 was anomalous when compared to the data from the seven other A/C in our database⁷. After discussions including NAVAIR, diagnosticians at the Original Equipment Manufacturer (OEM) and NAWCAD Trenton, the decision was made to down the A/C and notify the associated squadron of the downing due to a potential aft transmission failure.

At the time that the A/C was downed, the time on this particular aft transmission was 830.6 hours. The Time Between Overhaul (TBO) for this type of transmission is 900 hours. Typically, a 10% life waiver is requested, and the aft transmission will fly for approximately 990 hours between overhauls. The data collection causing the anomalous response was started at an aft transmission time of 710.7 hours and continued for approximately 20 hours at Marine Corps Air Station El Toro where the A/C had been attached to Marine Medium Helicopter Squadron (HMM) 764. Recent to the downing, the A/C had been deployed ferrying troops ship-to-shore during Operation Kernal Blitz '95.

After downing, the transmission was to be removed and sent to Naval Depot (NADEP) Cherry Point for Engineering Inspection (EI) to determine the status of the transmission components. If the A/C had not been downed, it could have been expected to fly for another 159.4 hours since there were no other

⁶ Note that for each network trained, the centers of the initial data clusters differed. Further, the weights for each training session were randomly initialized prior to convergence. Thus, none of the networks after training were identical.

⁷ The Flight Test Engineering Group at Pax River had been responsible for the data collection while NAVAIR 4.2.2 had program oversight regarding the aft transmission neural network effort.

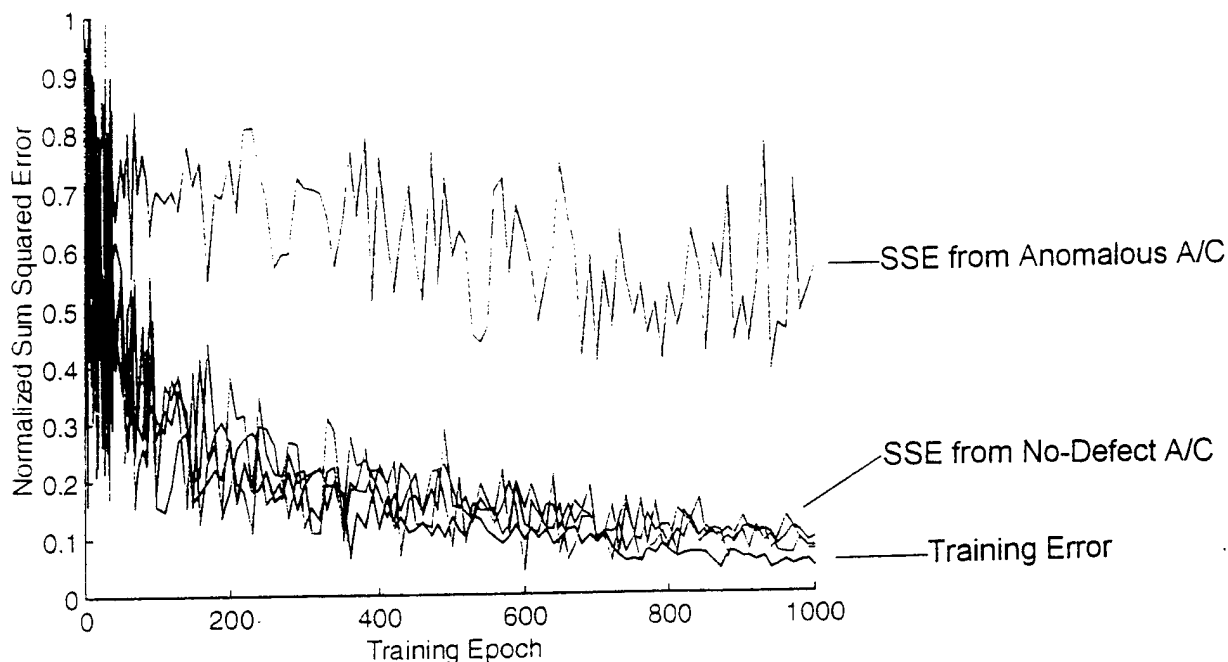


Figure 11. Trend in the SSE as a function of training epoch. The SSEs have been normalized by the maximum sum-squared error encountered during the training run. The SSE for the training data set is plotted as a solid line. The SSE for the three A/C in the cross-validation data set that did not have an anomalous response are shown as the lightly shaded lines while BuNo 156432 is shown as the darkly shaded line.

indications of a potential problem with the transmission. Periodic Spectral Oil Analysis Program (SOAP) burns had been passed with no anomalies indicated, and the squadron reported no special maintenance actions for this A/C. Additionally, no unusual behavior from the transmission had been reported by the crew. To the date of downing, the only indication that there was a potential problem with the transmission had been the anomalous response of the RBF networks.

5.7 RESULTS OF THE ENGINEERING INSPECTION (EI)

The EI on transmission S/N A9-328 was performed by the NADEP Cherry Point personnel who are responsible for performing the standard 900-hour rebuilds on the CH-46 transmission. When the transmission cases were cracked, separating mix box from the aft transmission, the spiral bevel ring and pinion gear were found to have obvious and considerable damage. Both mating surfaces of the gears had considerable wear to the point of cold metal working on their meshing surfaces. The wear was evenly distributed for all teeth of the gears and a considerable portion of the mesh was worn (please refer to figures 12 and 13). The wear was so extensive that depot personnel called it severe galling. Depot personnel also said that if this was not the worst, then it was the second-worst type of this damage that they had seen.

As the EI progressed, the collector gear was found to be in good condition with a minimal and normal amount of wear. Intervening bearings and shafts were also found to be in good condition and at proper preload and alignment. No other components were found to be damaged until the helical input pinions were removed from the mix box case. When the input pinions were removed, the starboard input pinion was found to have a localized chip or gouge and associated wear surrounding the gouge (figure 14). It is believed that this gouge was caused by debris passing through the mesh.

To summarize the EI, the transmission was anomalous and damaged. Two out of the three areas we had identified as potential problems were found to have defects. The defects were non-trivial and caused rejection of the parts by NADEP Cherry Point. In particular, the spiral bevel gear had a very high degree of wear and was new at the time that this transmission was last rebuilt. Such gears typically last two to three rebuilds.

5.8 AUTO-ASSOCIATOR NEURAL NETWORK

Once the transmission had proved defective, the in-flight defect data were used to validate and demonstrate the utility of a new type of neural network. We had recently been working with an Auto-Associator (AA) or encoding network for novelty detection. This type of network is a three-layer, feed-forward network that attempts to map the output onto the input using a reduced number of hidden or middle layer nodes (figure 15). In this type of architecture, the input layer and output layer have the same number of nodes in order to facilitate quantifying the success in mapping the output onto the input. The middle layer has a reduced number of nodes when compared to the input and output layers. The purpose of the hidden layer is to encode the essential character of the training class by use of a reduced set of information about the training class.

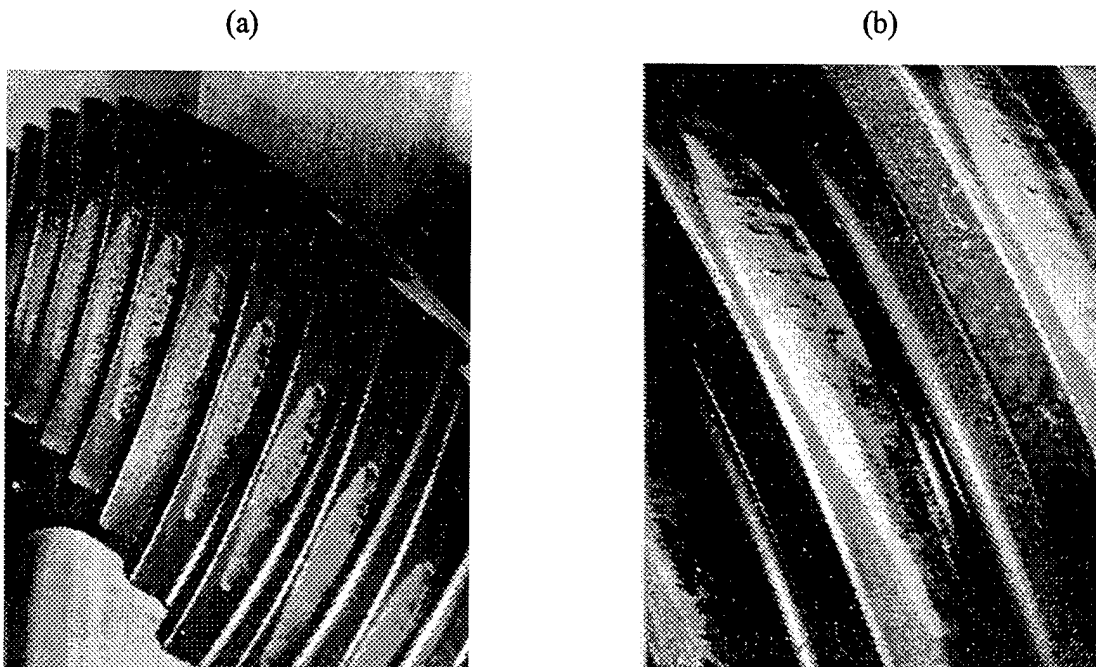


Figure 12. Damage to spiral bevel ring gear (a) general and (b) close-up. Damage occurs around the entire gear and across a significant portion of the mesh. Galling has worn through the black metal oxide coating and is close to cold metal working.



Figure 13. Damage to spiral bevel input pinion gear (a) general and (b) close-up. The damage is symmetric to that of the mating spiral bevel ring gear. Again, the galling has worn through the black metal oxide coating and is close to cold metal working.

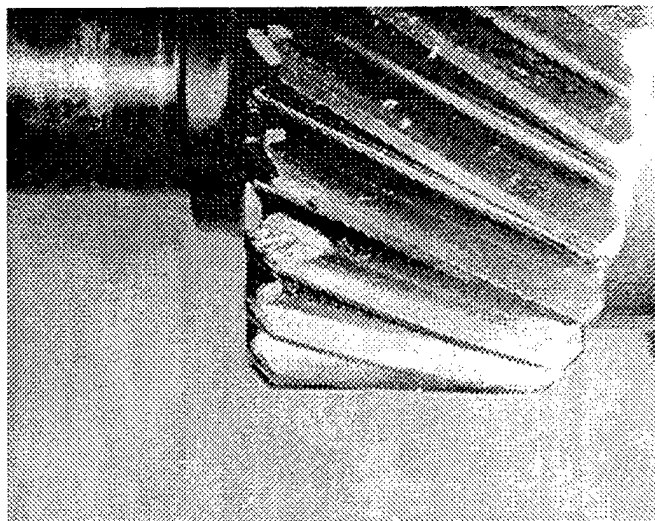


Figure 14. Damage to starboard helical input pinion gear. Gouge or chip is localized to a single tooth near the edge of the mesh.

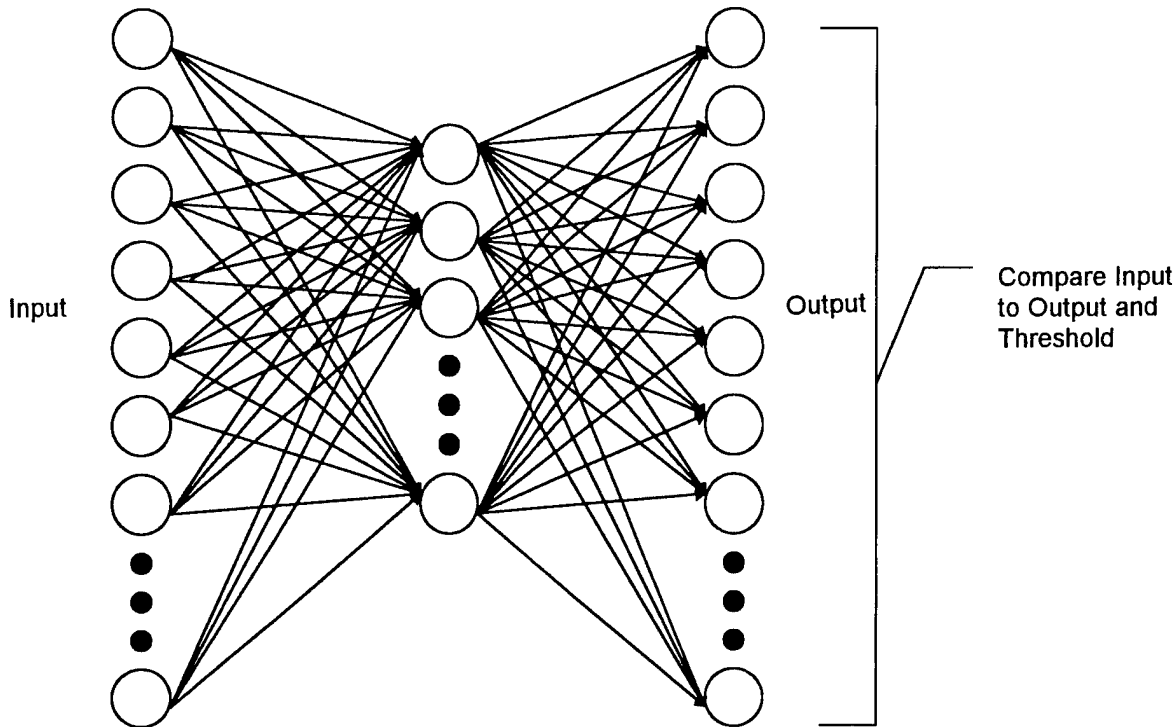


Figure 15. Architecture of a typical AA network. This network has three layers that are fully interconnected. The hidden or middle layer is of a reduced number of nodes when compared to the input or output layer in order to encode the essence of the training class of data. The input and output layer have the same number of nodes. After training, the output of a feature vector is compared to the input and the RMS error is calculated between the two. The error is then compared to a threshold. If the error is greater than the threshold, the input is considered novel.

This type of network differs significantly from typical pattern classifiers in that only "positive" data corresponding to a single class of data is used to train the network. During training, the network learns to map the output onto the input by encoding the essential character of the training class into the hidden layer. This is done by taking the output vector, reconstructed from the hidden layer, and comparing it to the input vector. A Root-Mean-Squared (RMS) error is then calculated from the difference of the input and output vectors. The interconnection weights are adjusted during the training process to reduce the RMS prediction error.

Once the network has been trained and a sufficiently small RMS prediction error achieved for the elements of the training data set, a threshold is set on the RMS error using cross-validation techniques. Typically, the threshold is set so that all members of the training class of data fall below the threshold while RMS prediction errors calculated from vectors corresponding to other classes of cross-validation data fall above the threshold. The network is then ready to be used in a specific application. When the network is employed, a feature vector never seen before by the network is passed in as input. The middle layer attempts to reconstruct the input using the mapping encoded in the layer during training. The input and output feature vectors are then compared and the RMS prediction error between the two is calculated. A large RMS error implies that the feature vector, for whatever reason, is novel or different than the class of data that the network has been trained on. Note that the network does not classify this vector to a specific class of data, it just denotes the feature vector as different or not a member of

different or not a member of the class of data that the network was trained on. Essentially, the network will tell you *when* something is novel not *why* it is novel.

In the AVDS gearbox application, the AA network had been trained using the original (including the in-flight data from the four good A/C) RBF training set of no-defect data. The test-rig defect data had then been used to cross validate the network and verify the threshold setting for RMS prediction error. The original cross-validation data set that caused the RBF network to indicate an anomaly was then presented to the AA network. The results of this test are shown in figure 16.

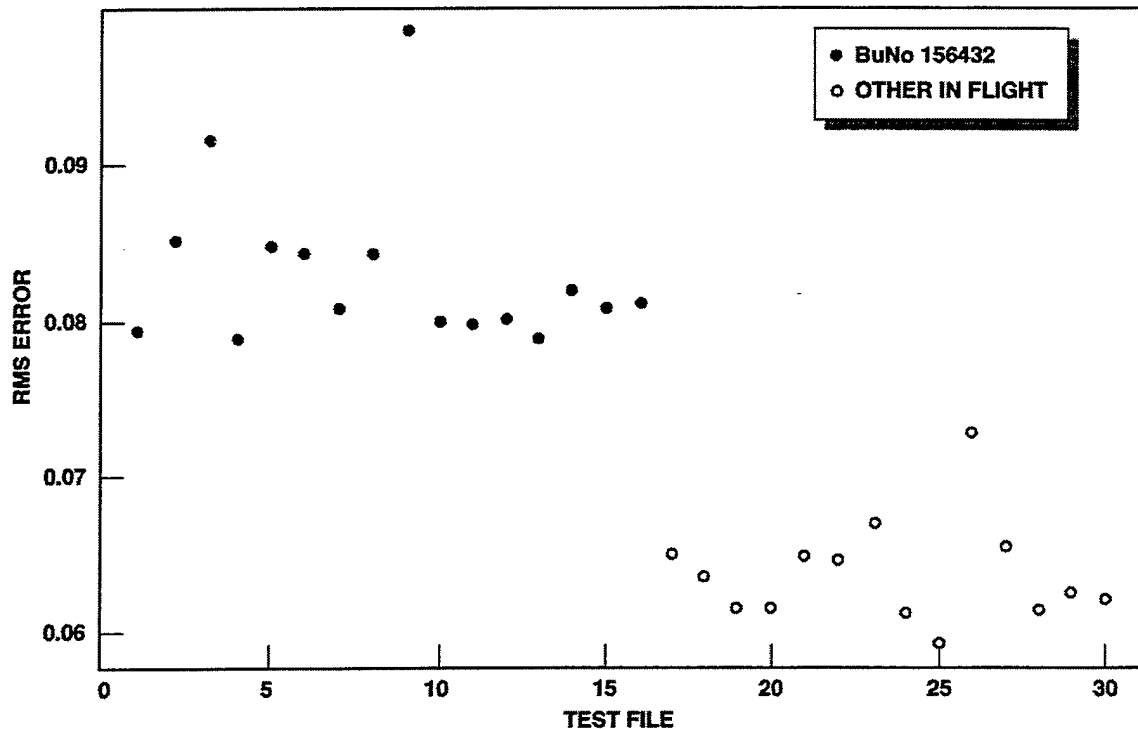


Figure 16. Blind testing RMS prediction error vs. feature vector run for the trained AA network. The feature vectors from A/C Bu No. 156432 are shown in black while the feature vectors from the three other A/C in the cross-validation test set are shown in white. The prediction error threshold is set at approximately 0.75.

The figure shows that AA net correctly indicated the anomaly and did not false alarm for the three other A/C in the cross-validation test set. The AA net was able to correctly identify an anomalous condition consisting of multiple defects in an A/C when trained using only positive no-defect data. The AA net also demonstrated the capability to correctly assess the condition of multiple A/C with which the network had no previous experience.

6 NEURAL NETWORK PROCESSOR (N3P) FLIGHT TRIALS

6.1 INTRODUCTION

The AVDS CH-46E Flight Trials demonstrated the potential of the N3P approach to transmission diagnosis. During the course of the trials, there were no false alarms and all classifications of the status of the aft main transmission of A/C 692 were to a healthy transmission. All classifications were made by the N3P without having to baseline the system to the transmission of A/C 692. The potential of the approach to identify those situations where sensor/data flaws bring into question the reliability of the results was also demonstrated. Unfortunately, the minimal quantity of data collected from a single A/C during the trials only shows the potential of the approach and does not provide a critical, defensible, quantified metric on the performance of the system.

For the analysis of the flight trials, it will be of benefit to quickly review the installation and communications of the N3P systems. The N3P hardware was installed in an avionics-hardened VMEbus chassis for placement in the A/C. A Teledyne-supplied signal conditioning card, a Teledyne-supplied phase lock loop and oscillator card, and a flight regime processor/communications card were also installed in the VMEbus chassis (the installation is shown in figures 17 and 18). The N3P would receive analog conditioned signals from the aft main transmission accelerometers and an analog conditioned rotor position shaft tachometer signal via the Teledyne-supplied cards. The N3P would also receive Data Acquisition Flight Conditions (DAFC) from the MDAU via the flight regime processor card at a 4-Hz rate for use in data processing. In turn, the N3P would provide a status to the MDAU at a 1-Hz rate. Note that the health status of the transmission is not necessarily calculated at a 1-Hz rate. In the case where all conditions were present so that the N3P could analyze and classify the condition of the transmission, this calculation took approximately 3 seconds. The intervening status messages that went to the MDAU would be latched to the previous status in the output buffer during the calculation. Thus, the total number of status messages does not necessarily represent on a one-to-one basis the amount of data (in seconds) that was processed by the system.



Figure 17. Front view of the VMEbus installation in A/C 692. The VMEbus chassis containing the N3P is located in the lower portion of the equipment rack. The OQAR that logged the N3P status message is the rectangular black box shown in the upper left of the equipment rack.

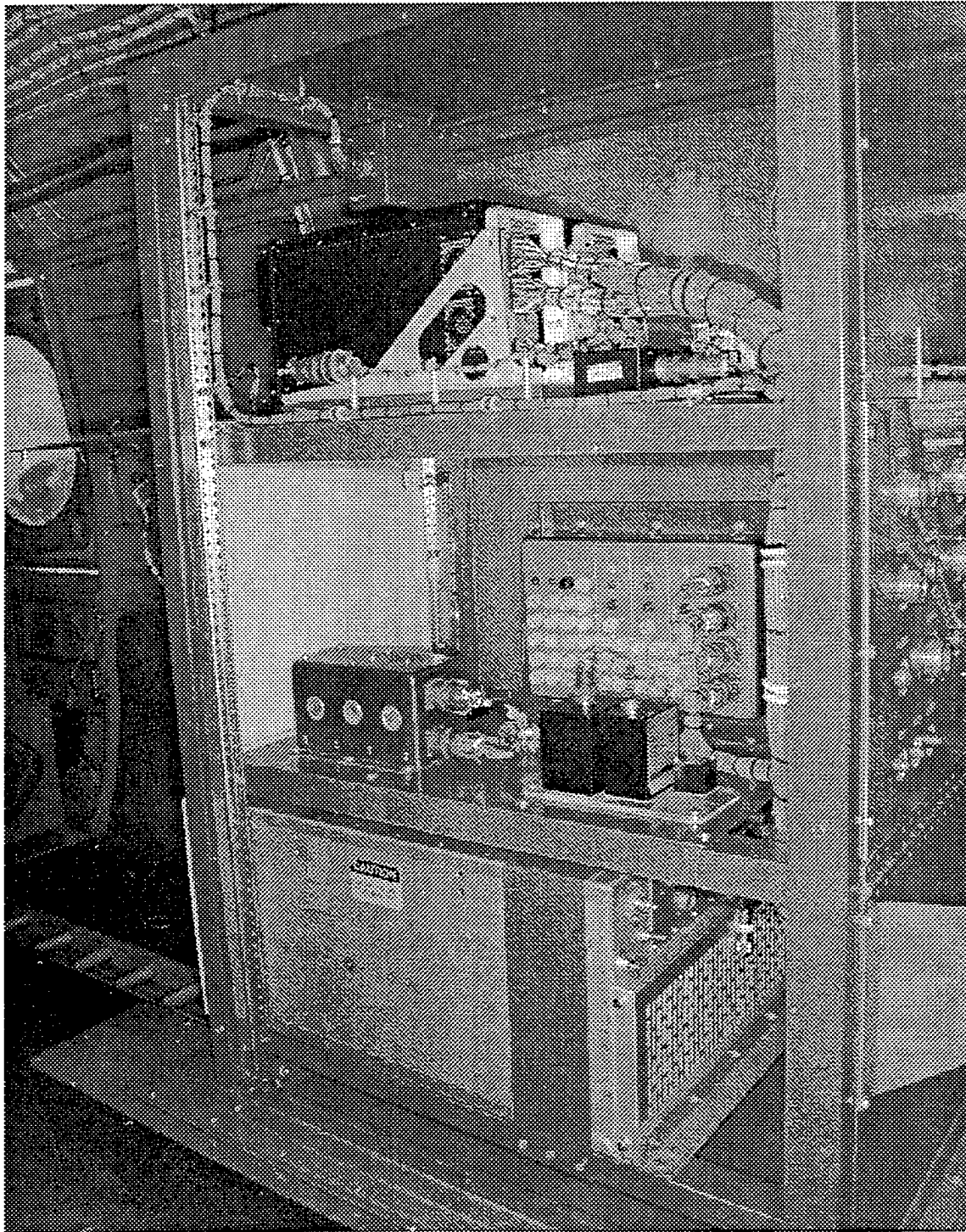


Figure 18. Side view of the VMEbus installation.

6.2 PROCESSING THE RESULTS OF THE TRIAL

Once the N3P status message was received by the MDAU via the flight regime processor card, it was then sent to the OQAR for data logging. The status message consisted of four fields; the report indicator, fault location, fault severity, and DAFC. A report indicator of value

1 to 3 indicates a healthy transmission (for a breakdown of the possible report indicators, please see table 12). A report indicator of 4 to 8 indicates a faulty or anomalous transmission.

Table 12. Report indicator breakdown.

Report Indicator Value	Description
0	No Report At This Time
1	Good (Radial Basis Function)
2	Good (Auto-Associator)
3	Good (Radial Basis Function and Auto-Associator)
4	Anomaly (Auto-Associator)
5	Anomaly (Radial Basis Function and Auto-Associator)
6	Anomaly & Fault
7	Fault (Radial Basis Function)
8	Fault (Radial Basis Function and Auto-Associator)
9	Neural Network Confidence Failure

The fault location and fault severity fields were never used during the trials since no faults or anomalies were indicated during the course of the trials. The DAFCs in which the N3P attempted to process data and calculate the status of the transmission are listed in table 13. Note that the DAFC had to have been stable for approximately 15 seconds prior to and approximately 15 seconds after the calculation in order for the health status of the transmission to be output from the N3P to the MDAU.

After the flight, the N3P status messages on the OQAR disc could be browsed and converted for analysis using the Teledyne-supplied FLIDRAS software. The converted files then had to be edited prior to importation into the N3P groundstation for display. During and after the trials, the process of downloading the N3P messages turned out to be a convoluted and time consuming process that increased the time latency of the results and made troubleshooting and analysis a difficult and tiresome process.

In the N3P groundstation, the analysis of the results was limited to a sequence of histogram displays used to analyze the results and an overall status of the aft main transmission and N3P system for the flight. The messages could be broken down to display histograms of the status message sorted by types of report indicator (figure 19), confidence indicated in the answer (figure 20), and subsets of the answer corresponding to no fault and faults/anomaly.

Histograms of the DAFCs received from the flight Regime Processor could also be displayed in the groundstation. The interim message file could also be manually examined in the groundstation to develop trend information and to select subsets of the flight for analysis and display.

Table 13. DAFC breakdown.

DAFC Value	Description
0	Flat pitch (27% torque)
1	Light on wheels (50% torque)
2	Hover in ground effect (70% torque)
3	Hover out of ground effect (75% torque)
4	750 feet per minute descent (30% torque)
5	60 knots indicated air speed (45% torque)
6	80 knots indicated air speed (40% torque)
7	100 knots indicated air speed (60% torque)
8	120 knots indicated air speed (80% torque)
9	140 knots indicated air speed (100% torque)
10	Spare (not currently used)
.	
.	
.	
13	Spare (not currently used)
14	Unknown on ground
15	Unknown in flight

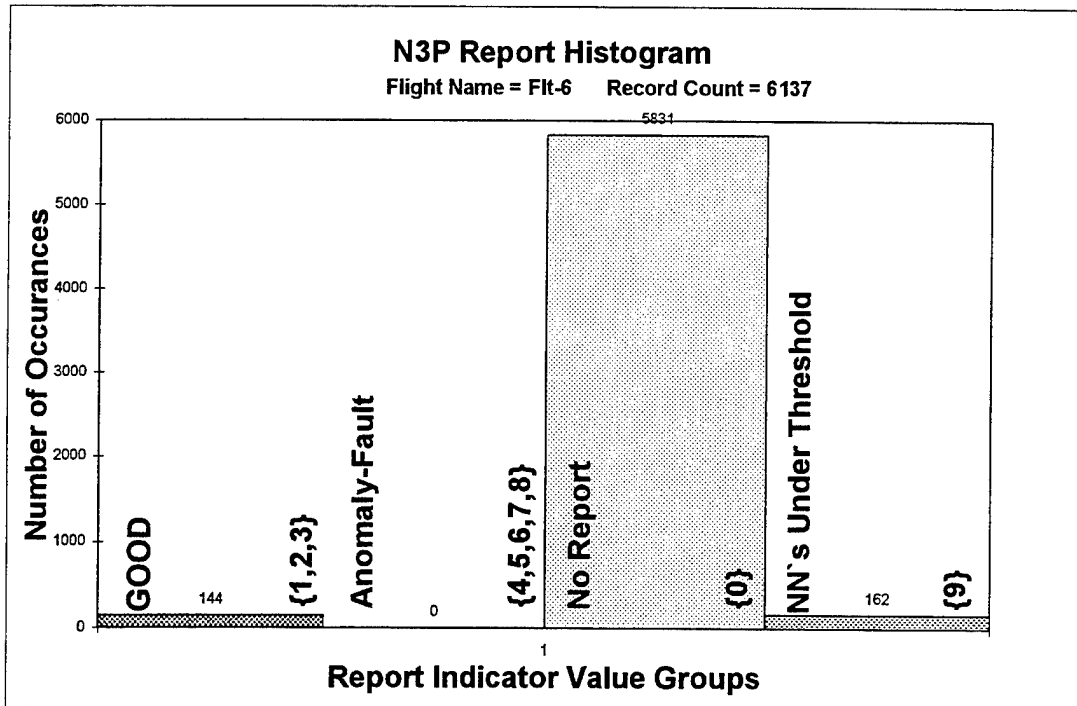


Figure 19. Sample histogram from Flight N3SHA-6 showing distribution report indicators broken down into fields of good transmission, anomalous-faulty transmission, no report at this time, and network confidence failures. During this flight, multiple sensor/wiring failures related to loose connectors and an accelerometer backing off its mounting bolt caused network confidence failures.

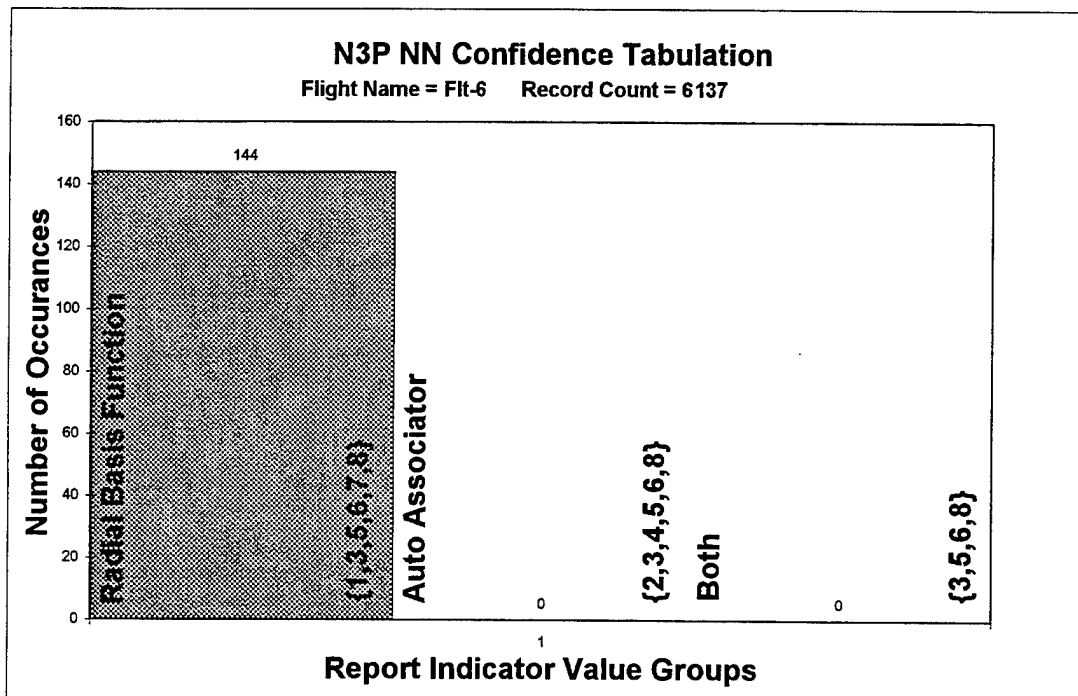


Figure 20. Sample histogram from Flight N3SHA-6 showing confidence indicated in the report indicators (figure 19). In this case, the majority of the no-fault answers provided by the RBF networks had marginal confidence while all answers provided by the Auto-Associator networks and some of the answers by the RBF networks had a relatively low confidence.

6.3 PROCESSED FLIGHT TRIALS DATA

During the course of the trials, there were 18 dedicated flights and one scripted piggy back flight (the flight to and from the USS *Saipan* LHA2). For the first flight, the N3P system had a hardware failure that prevented the system from booting and operating during the flight. The N3P system was not installed aboard the A/C during the second flight due to troubleshooting of the hardware. For the remaining 17 flights, at least some data from each flight are available for analysis (table 14). It is worth emphasizing that these data were derived without baselining the system to the transmission of 692, and that the N3P software operated without flaw during the trials. Two software changes were made during the trials, around Flight N3SHA-13, in an attempt to aid in troubleshooting the DAFC calculation and sensor failures, but these changes did not affect the diagnostic or performance characteristics of the N3P. One software change allowed time code to be written to exabyte tapes to help in troubleshooting the DAFC calculations while the other change stored an output file on the hard disk to allow "quick look" troubleshooting of the sensors and wiring.

Table 14. Report indicators broken down by flight. Flight N3SHA6-8 was particularly plagued with multiple sensor failures including bad microdot connections, accelerometers backing off mounts, problematic patch panel connections, intermittent signal saturation, etc.

FLIGHT	No Report Indicator	Good Indicator	Confidence Failure	Anomaly/Fault
N3SHA -3	5993	129	0	0
N3SHA -4	3864	240	0	0
N3SHA -5	3356	234	0	0
N3SHA -6	5831	144	162	0
N3SHA -7	2060	162	138	0
N3SHA -8	5806	114	60	0
N3SHA -9	5282	570	0	0
N3SHA -10	5947	779	0	0
N3SHA -11	5375	479	0	0
N3SHA -12	4631	334	0	0
N3SHA -13	5190	996	0	0
N3SHA -14	1970	381	0	0
N3SHA -15	6708	381	0	0
N3SHA -16	6157	144	0	0
N3SHA -17	5429	415	0	0
N3SHA -18	2525	807	0	0
N3SHA Saipan	4400	162	0	0
Total	80524	6471	360	0

The majority of no report at this time indicators encountered during the trials were due to development and refinement of the DAFC calculation and limits. Many of the remaining no report indicators and the network confidence failure indicators are due to sensor failures, connector failures, cabling failures, and equipment failures. Under the operating conditions of the trials, the A/C spent a fair amount of time transitioning between DAFCs and in-flight regimes that were not covered by the DAFCs used by the N3P. This normal operation accounts for the remainder of the no report indicators.

From the above data, two trends are important to note. First, over the course of the trials the DAFC calculations did tend to improve with a corresponding rise in the number of report indicators that actually classified the health of the transmission. This trend is positive, but not monotonic. Secondly, right up to the end of the trials the installation was plagued by sensor and cabling associated failures. There were also several unscripted piggyback flights where the N3P was installed aboard the A/C, but due to DAFC troubleshooting and faulty SCSI drive cabling, there is no N3P data available for analysis from these flights.

6.4 FLIGHT TRIALS CONCLUSIONS

During the course of the trials, the N3P system was up and running for a total of 24.3 hours. The N3P system provided 1.9 hours of status reports containing a condition identification other than no report at a 1-second interval rate. Of these 1.9 hours, 1.8 hours were actual classifications as to the health of the transmission. This 1.8 hours of no fault transmission classifications corresponds to 7.8% of the time that the N3P was up and running. The actual amount of data that were used to classify the transmission as to a no-fault status for this 7.8% of N3P uptime was 0.60 hours or 36 minutes.

There were no false alarms recorded during the course of the trials. The total amount of time that comprised an opportunity for a false alarm was 1.9 hours. The underlying amount of data comprising this opportunity correspond to 0.63 hours or 37.8 minutes. This window of opportunity for false alarm corresponds to 2.6% of the total N3P uptime. It should be obvious that extraction of a meaningful false alarm rate from this minimal amount of data from a statistical sample of one transmission is not possible. Therefore, even though no false alarms were indicated during the trials this is not taken to indicate with any confidence that the N3P false alarm rate is zero. The final conclusion is that the trials were encouraging, but by no means a definitive measure of the performance of the N3P system.

7 PROGRAM CONCLUSIONS

The capability of the N3P system to perform diagnostics without baselining to a particular build of the given transmission was demonstrated in all three phases of the program. During the Test Stand portion of the program the neural networks were trained using data from one build of the transmission and blind-tested on data from a build separated by nine major maintenance actions. The networks successfully classified the "blind-build" as no fault. When the system was trained using in-flight data, only data from four of the available A/C were used during training. The data from the remaining four A/C were used to blind-test the system. There was no baselining of the system to this set of data. It was out of this set of data that the find in A/C 156432 came. As previously noted, the system had an anomalous response to A/C 156432 and successfully classified the other A/C as no fault. During the flight trials, the system was installed on A/C 692 without baselining. The system was able to correctly diagnose the condition of the A/C without using data from the A/C to "tune" the system to this A/C.

Being able to operate the system without baselining to a given A/C is a significant achievement and a valid goal for the program. Typically, it takes COTS systems on the order of 20 dedicated flight hours to baseline to a given A/C. This baselining must be performed for each system installed on every A/C. The baselining must be repeated if a major maintenance action is performed on the transmission of the A/C. As one can see, the baselining effort is by no means trivial for a significant fleet of A/C. In particular, the Marine Corps alone has on the order of 238 CH-46E A/C which would require approximately 4760 flight hours of dedicated baselining.

The N3P system has also been demonstrated in all three phases of the program to operate across a range of torques. For all phases, there were at least nine torque settings where the N3P was consistently shown to operate and assess the condition of the transmission. This is also a valid goal for the program since in a military environment the A/C are operated in a manner of conditions across a wide range of regimes. A diagnostic system should adapt itself to the manner in which the A/C will be flown since military flight operations can hardly be expected to adapt themselves to the requirements of a diagnostic system.

7.1 RECOMMENDATIONS

In light of the experience gained during the course of the program, the following recommendations are made:

- **Accelerometers**

The microdot connectors of the Endevco 7259A-10 accelerometers proved to be quite problematic during the inflight phase of the program and the course of the flight trials. There is a major reliability versus sensitivity/bandwidth tradeoff issue that should no longer be a concern with many of the accelerometers available today. Most current accelerometers have a more than sufficient sensitivity and bandwidth for diagnostic purposes that makes the tradeoff issue become one of reliability. Accelerometers should be chosen first on the basis of their reliability and durability in the helicopter avionics environment, with sensitivity and bandwidth secondary issues. It is also recommended that the signal path from the

accelerometers to the diagnostic system be made as simple as possible with as few intervening connectors as possible.

- **Accelerometer Mounts**

The accelerometer mounts should have their vibrational response characterized prior to use. If the response has resonances or characteristics that would tend to mask the vibration signature, an alternate mount should be designed and characterized.

- **Accelerometer Locations**

The location of the accelerometers should be chosen to maximize the visibility of the component(s) that the accelerometer is chosen to cover. Interference in the signal from other components and transmission case resonances should be minimized. As an example of signal interference, the AVDS accelerometer #8 signal was dominated by the vibration from the underbody oil scavenge pump. The signals from gear box accessories and other components were effectively masked by the pump signature. To adequately determine locations that maximize the visibility of monitored components, the accelerometer locations should be determined using an instrumented survey.

- **Data Quality Assurance**

On-line, automated data quality assurance should be available. This data quality assurance should be both passive and active. The types of quality assurance should be chosen to ensure that the data is free from defects and artifacts, but the implementation should be chosen so that ease of use and availability for real-time troubleshooting are maximized.

- **Flight Trials Performance Metrics**

The performance metrics used to assess performance should be well-defined and chosen in advance of the trials. The metrics should be unambiguous, easy to compute, and adequately characterize any performance tradeoffs related to the system. The metrics should also provide an apples-to-apples comparison between different systems that are being evaluated at the same time. This is especially true in the case where false alarm rates are a significant measure of the performance of the system. In particular, many COTS systems have a decision interval on a given component that is on the order of minutes. By contrast, the N3P has a decision interval on the complete assembly on the order of 3 seconds. In order to compensate for the differing decision intervals and component coverages, the false alarm rates calculated for the two types of systems would have to be time base normalized and component coverage normalized.

- **Flight Trials Design**

The design of the flight trials should be driven by the performance metrics used to quantify performance during the trials. The trials should be designed so that the metrics can be established and properly quantified. Design of the trials should also place the different systems under evaluation on an equal footing during the trials.

- **Representative Fleet Sample**

In order for the results of the trials to be statistically meaningful, the trials should encompass a greater period of time spread across a larger number of A/C. On each A/C, the systems should be evaluated for approximately 10% of the life of the transmission between rebuilds. The evaluation should take place on a representative sample of the A/C in the fleet. Approximately 5-10% of the A/C comprising the fleet should have systems installed and evaluated during the course of the trials.

- **Continued Development of Neural Network Diagnostic Techniques**

Test Stand, In-flight, and Flight Trials results suggest that the N3P approach appears to offer significant advantages over COTS technology, but that the approach requires further evaluation. Since the results to date have been so promising, continued development of the N3P system should be pursued. The system should be evaluated on a number of fleet A/C for a longer duration so that credible false alarm rate statistics can be established. This evaluation should take place prior to efforts to "productionize" the system.

APPENDIX A
AIR VEHICLE DIAGNOSTIC SYSTEM SIGNAL PROCESSING AND
CLASSIFICATION ROUTINES. BOARD-LEVEL IMPLEMENTATION.

Wed Apr 23 14:30:51 1997

```

/*****
 *
 * File: host.c
 * Program: host
 * OS: SunOS v4.1.1b
 * Target: Force 2ce (Host)
 * Execute: host apxprog
 * Link list: /usr/sky/bolt/host/1lb/skyload.a
 *
 * This program provides the framework for split application with shared
 * memory and interprocessor communication (IPC) using shared memory.
 *
 * Rev History: 15 Mar 95 - dkl
 * 19 Apr 95 - mep, modified interface for ipc shared mem
 * from ap0 -> apl,ap2,ap3. outputs are
 * now being processed by the host (Sun).
 * 5/17/95 (MEP) - modified to enable host control of data
 * block transfers to AP's 1-3. Interim
 * support for automatic matlab spectrum
 * processing.
 * 9/05/95 (MEP) - modified to add the second SKY and save
 * data to disk for data acquisition modes
 * direct to hard disk.
 * 3/27/96 (MEP) - add GIU interface and sockets
 * 6/26/96 (MEP, RCG) - added 100 node AA, normalize
 * threshold initialization
 *
 *****/
#define MAIN_PROG
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/time.h>
#include "adregs.h"
#include "usr/sky/bolt/inc/sky/skyprotos.h"
#include "hostshm.h"
#include "datahist.h"

#ifdef RUN_FOREVER
#define A2D
#define RUN_FOREVER
#define TAPE_INSTALLED
#endif
#define RUN_AVDS
#define A2D
#define PLAYBACK
#define TAPE_INSTALLED
#endif

/*****
 *
 * included facilities
 *
 *****/

```

```

#define SAVE_OUTPUTS
#define SAVE_TO_DISK
#define ICS_VSB_MODE
#define NUM_BITS_BITS12
#define RBF_INSTALLED
#define AA_INSTALLED
#define SAVE_OA_ERRORS
#define BOARD_1_SKY
#define FULL_OUTPUTS
#define OUTPUT_TO_CONSOLE
#define RING_BELL_ON_ERROR
#define SAVE_MESSAGE_DATA

/*****
 *
 * excluded facilities
 *
 *****/
#define FALSE

#define TIMECODE_INSTALLED
#define MATLAB_DISPLAY
#define PRINT_HOST_STATUS
#define INTERNAL_CLOCK
#define ICS_VME_MODE
#define PRINT_WARNINGS
#define PRINT_IC3_STATUS
#define PRINT_PO_OUTPUTS
#define PRINT_PI_OUTPUTS
#define SINGLE_THRESHOLD

#endif

```

```

/*****
 *
 * forward declarations
 *
 *****/

void apSignalCatcher();
int apWaitForSignal();
void timeSignalHandler();
int setupICS110();
int checkICSStatus();
int mvPOOutputs();
int mvPIOutputs();
void printQAEError();
unsigned long *mapVME32();
void PLLReset();
int RbfAcceptDetermine();
int AAAcceptDetermine();
void printAnswerText();
int readTimeCode();
void delay();
void sendMinMax();
void sendQARes();

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

```

racocon/mn/avds/v5.0/sky/host/host.c

Wed Apr 23 14:30:51 1997

```

* local constants
*
\*****
#define TITLE_CNT 10
#define MAX_CHAN_ERROR 3
#define MAX_CHAN_ERROR_RESET_COUNT 6
#define TACH_ERROR_RESET_COUNT 4
#define REF_ERROR_RESET_COUNT 6
#define ICS_VSB_FIFO_UPPER 0xb800
#define SAMPLE_RATE 100.0 /* in KHz */
#define TIMEOUT_IN_SECS 0
#define TIMEOUT_IN_USECS 250000
#define COUNTS_PER_SECOND 4
#define MM_MEM_ADDRESS 0x60000000
#define MM_MEM_SIZE 0x08000000
#define HOST_STATUS_RATE 10 /* 250 ms per tick */
#define CYCLE_RATE 20 /*
/*****
* module level variables
*
\*****
int waitFlag;
int waitForthTime;
int ics110;
int cycle;
int fdout = 0;
FILE *con = NULL;
FILE *fimage = NULL;
int QAFram=0, netFrame=0;
unsigned long *vmeMM;
/* error variables */
int QAErrors = 0;
int ChanErrors = 0;
int ErrorChan;
int ErrorsDuringRun = 0;
int MaxChanResets = 0, tMaxChanErrors = 0, MaxChanErrors = 0;
int TachResets = 0, tTachErrors = 0, TachErrors = 0;
int RefResets = 0, tRefErrors = 0, RefErrors = 0;
int lastICSstat = 0;
int day, hour, minute, second, millisec;
int PLLSock;
#define MOTIF_DISPLAY
int displaySock;
int dspCmd;
#define MOTIF_DISPLAY
int dspCmd;
#define MOTIF_DISPLAY
char messageBuffer[32];
#endif
float Aamln = 0.02;
float Aamax = 0.09;
float Aathresh = 0.052;

float RbNetQ[RBF_AVERAGE_COUNT][NUM_FAULTS];
float AANetQ[AA_AVERAGE_COUNT];
float minAcceptValue[NUM_FAULTS] = {
    0.39, /* node 0 */
    0.45, /* node 1 */
    0.53, /* node 2 */
    0.37, /* node 3 */
    0.31, /* node 4 */
    0.4, /* node 5 */
    0.54, /* node 6 */
    0.5, /* node 7 */
    0.6, /* node 8 */
};
float minAcceptDiff[NUM_FAULTS] = {
    0.09, /* node 0 */
    0.1, /* node 1 */
    0.25, /* node 2 */
    0.09, /* node 3 */
    0.08, /* node 4 */
    0.1, /* node 5 */
    0.2, /* node 6 */
    0.26, /* node 7 */
    0.37, /* node 8 */
};

int NoResults = 0xffff; /* 0,15,15,15 */
int Rbf_u_AA_u = 0xffff; /* 9,15,15,15 */
int Rbf_a_AA_u[] = {
    0xffff, /* 1,15,15,15 */
    0x0307, /* 7, 0, 3, 0 */
    0x0317, /* 7, 1, 3, 0 */
    0x0327, /* 7, 2, 3, 0 */
    0x0437, /* 7, 3, 4, 0 */
    0x0447, /* 7, 4, 4, 0 */
    0x0457, /* 7, 5, 4, 0 */
    0x0667, /* 7, 6, 0, 0 */
    0x0377, /* 7, 7, 3, 0 */
};

int Rbf_u_AA_a[] = {
    0xffff2, /* 2,15,15,15 */
    0x00f4, /* 4,15, 0, 0 */
};

int Rbf_a_AA_a[] = {
    0xffff3, /* 3,15,15,15 */
    0x0308, /* 8, 0, 3, 0 */
    0x0318, /* 8, 1, 3, 0 */
    0x0328, /* 8, 2, 3, 0 */
    0x0438, /* 8, 3, 4, 0 */
    0x0448, /* 8, 4, 4, 0 */
    0x0458, /* 8, 5, 4, 0 */
    0x0668, /* 8, 6, 0, 0 */
    0x0378, /* 8, 7, 3, 0 */
    0xffff5, /* 5,15,15,15 */
    0x0306, /* 6, 0, 3, 0 */
    0x0316, /* 6, 1, 3, 0 */
    0x0326, /* 6, 2, 3, 0 */
    0x0436, /* 6, 3, 4, 0 */
    0x0446, /* 6, 4, 4, 0 */
    0x0456, /* 6, 5, 4, 0 */
};

```

racon/mnt/avds/v5.0/sky/host/host.c

`raccoon:/mnt/avds/v5.0/sky/host/host:c`

Wed Apr 23 14:30:51 1997

```

*
\*****
main (argc, argv, argp)
int argc;
char *argv[];
char *argp[];
{
    struct timeval timectl;
    int i;
    int AP0pid;
    int AP1pid;
    int AP2pid;
    int AP3pid;
    int P1AP0pid;
    int P1AP1pid;
    int P1AP2pid;
    int retval;
    char *string;
    char dBuff[1024];
    int startFrame;
    int numFrames;
    #ifdef PRINT_PO_OUTPUTS
    int P0printFlag = DISPLAY;
    #else
    int P0printFlag = SUPPRESS;
    #endif
    #ifdef PRINT_P1_OUTPUTS
    int P1printFlag = DISPLAY;
    #else
    int P1printFlag = SUPPRESS;
    #endif
    #ifdef A2D
    int recentReset = 0;
    #endif
    #ifdef MATLAB_DISPLAY
    int matfd;
    char frameStr[10];
    #endif
    #ifdef MOTIF_DISPLAY
    int MinMaxIndex = 1e32;
    int displayBuff;
    #endif
    #ifdef RUN_FOREVER
    numFrames = atoi(argv[8]);
    #endif

    #ifndef FILEMODE
    /* connect a socket to the mv166 for command processing */
    P1Lsock = createConnectionSocket("woodchuck", 2045);
    #endif

    #ifdef MOTIF_DISPLAY
    /* connect a socket to the Display process */
    displaySock = createConnectionSocket("agouti", 2050);
    /* send neural net threshold data to display */
    delay(10000);
    SocketWrite(displaySock, minAcceptValue, sizeof(float)*(NUM_FAULTS));
    SocketWrite(displaySock, &Aathresh, sizeof(float));
    #endif
}

#ifdef TAPE_INSTALLED
delay(1000000);
#endif
#define A2D
SocketWrite(P1Lsock, "trst", 5);
#else
startFrame = atoi(argv[8]);
numFrames = atoi(argv[9]);
SocketWrite(P1Lsock, "play", 5);
delay(1000000);
SocketWrite(P1Lsock, &startFrame, 4);
#endif

/* Set interrupt and shared memory (4 pages @ 4 KB) */
string = "BOLTpolltime=0";
retval = putenv (string);
printf ("Host: putenv of BOLTpolltime 0 returned %d\n", retval);
fflush (stdout);

string = "BOLTuserdata=7";
retval = putenv (string);
printf ("Host: putenv of BOLTuserdata 1 returned %d\n", retval);
fflush (stdout);

/* Set nexus and AP routine; Attach SKYbolt processes */
/* API */
string = "BOLTdev=0011";
retval = putenv (string);
printf ("Host: putenv of BOLTdev 0_0_1_1 returned %d\n", retval);
fflush (stdout);
argv[1][2] = '1';

AP1pid = parse_and_boltattach (argc, argv, argp);

if (AP1pid <= 0) {
    printf ("Host: Error on SKY attach, status = 0x%x\n", AP1pid);
    fflush (stdout);
    exit (1);
}

/* Unblock SKYbolt signals from API */
unblock_bolt_signals();

/* Setup signal catcher to API process */
host_signal (SIGUSR1, (void *) apSignalCatcher);

/* wait for API to signal */
if (apWaitForSignal() == 0) {
    printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
}

```

raccoon/mn/avds/v5.0/sky/host/hostc

racoon:/mnt/avds/v5.0/sky/host/host.c

Wed Apr 23 14:30:51 1997

```

retval = putenv (string);
printf ("Host: putenv of BOLTdevdata 1 returned %d \n", retval);
fflush (stdout);
argv[1][2] = '0';
AP0pid = parse_and_boltattach (argc, argv, argv);
if (AP0pid <= 0) {
    printf ("Host: Error on SKY attach, status = 0x%x \n", AP0pid);
    fflush (stdout);
    exit (1);
}
/* Unblock SKYbolt signals from AP0 */
unblock_bolt_signals();
/* Setup signal catcher to AP0 process */
host_signal (SIGUSR1, (void *) apSignalCatcher);
/* wait for AP0 to signal */
if (apWaitForSignal() == 0) {
    printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
    exit (1);
}
/* Unblock SKYbolt signals from P1/AP0 */
unblock_bolt_signals();
/* Setup signal catcher to AP0 process */
host_signal (SIGUSR1, (void *) apSignalCatcher);
/* wait for AP0 to signal */
if (apWaitForSignal() == 0) {
    printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
    exit (1);
}
/* Get AP0 shared memory address */
hostShmP1AP0 = (struct rbfData *)get_share_mem_addr();
hostShmP1AP0->PrintFlag = P0PrintFlag;
printf ("Host: P1/AP0 Shared memory address = 0x%x \n", hostShmP1AP0);
fflush (stdout);
#ifdef AA_INSTALLED
/* API */
string = "BOLTdev=1011";
retval = putenv (string);
printf ("Host: putenv of BOLTdev 1_0_1_1 returned %d \n", retval);
fflush (stdout);
argv[1] = "P1ap1prog";
P1AP1pid = parse_and_boltattach (argc, argv, argv);
if (P1AP1pid <= 0) {
    printf ("Host: Error on SKY attach, status = 0x%x \n", P1AP1pid);
    fflush (stdout);
    exit (1);
}
/* Unblock SKYbolt signals from P1/API */
unblock_bolt_signals();
/* Setup signal catcher to API process */
host_signal (SIGUSR1, (void *) apSignalCatcher);
/* wait for API to signal */
if (apWaitForSignal() == 0) {
    printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
    exit (1);
}
}

```

```

retval = putenv (string);
printf ("Host: putenv of BOLTdevdata 1 returned %d \n", retval);
fflush (stdout);
argv[1][2] = '0';
AP0pid = parse_and_boltattach (argc, argv, argv);
if (AP0pid <= 0) {
    printf ("Host: Error on SKY attach, status = 0x%x \n", AP0pid);
    fflush (stdout);
    exit (1);
}
/* Unblock SKYbolt signals from AP0 */
unblock_bolt_signals();
/* Setup signal catcher to AP0 process */
host_signal (SIGUSR1, (void *) apSignalCatcher);
/* wait for AP0 to signal */
if (apWaitForSignal() == 0) {
    printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
    exit (1);
}
/* Get AP0 shared memory address */
hostShmP0AP0 = (struct IOControl *)get_share_mem_addr();
printf ("Host: AP0 Shared memory address = 0x%x \n", hostShmP0AP0);
fflush (stdout);
#ifdef BOARD_1_SKY
/* PLATFORM 1 */
/* RBF INSTALLED */
/* AP0 */
/* Always do AP0 last in boltattach sequence */
string = "BOLTdev=1001";
retval = putenv (string);
printf ("Host: putenv of BOLTdev 1_0_0_1 returned %d \n", retval);
fflush (stdout);
argv[1] = "P1ap0prog";
P1AP0pid = parse_and_boltattach (argc, argv, argv);
if (P1AP0pid <= 0) {
    printf ("Host: Error on SKY attach, status = 0x%x \n", P1AP0pid);
    fflush (stdout);
    exit (1);
}
}

```

racoony/mnt/avds/v5.0/sky/host/host.c

Wed Apr 23 17:30:51 1997

```

    printf ("Host: Success on SKY attach, pid = 0x%x \n", PIAP1pid);
    fflush (stdout);
    /* Get AP1 shared memory address */
    hostShmPIAP1 = (struct aaData *)get_share_mem_addr();
    hostShmPIAP1->PrintFlag = PPrintFlag;
    printf ("Host: P1/AP1 Shared memory address = 0x%x \n", hostShmPIAP1);
    fflush (stdout);
    #ifdef MOTIF_DISPLAY
    /* AP2 */
    string = "BOLTuserdata=65";
    retval = putenv (string);
    printf ("Host: putenv of BOLTuserdata 1 returned %d \n", retval);
    fflush (stdout);
    string = "BOLTdev=1021";
    retval = putenv (string);
    printf ("Host: putenv of BOLTdev 1_0_2_1 returned %d \n", retval);
    fflush (stdout);
    argv[1] = "Plap2prog";
    PIAP2pid = parse_and_boltattach (argc, argv, argc);
    if (PIAP2pid <= 0) {
        printf ("Host: Error on SKY attach, status = 0x%x \n", PIAP2pid);
        fflush (stdout);
        exit (1);
    }
    /* Unblock SKYbolt signals from P1/AP2 */
    unblock_bolt_signals();
    /* Setup signal catcher to AP2 process */
    host_signal (SIGUSR1, (void *) apSignalCatcher);
    /* wait for AP2 to signal */
    if (apWaitForSignal() == 0) {
        printf ("Host: timeout occurred waiting for SIGUSR1 signal.\n");
        fflush (stdout);
        exit (1);
    }
    printf ("Host: Success on SKY attach, pid = 0x%x \n", PIAP2pid);
    fflush (stdout);
    /* Get AP2 shared memory address */
    hostShmPIAP2 = (struct imageData *)get_share_mem_addr();
    hostShmPIAP2->PrintFlag = PPrintFlag;
    printf ("Host: P1/AP2 Shared memory address = 0x%x \n", hostShmPIAP2);

    fflush (stdout);
    #endif
    hostShmP0AP1->PrintFlag = PPrintFlag;
    hostShmP0AP2->PrintFlag = PPrintFlag;
    hostShmP0AP3->PrintFlag = PPrintFlag;
    /* Setup timer and hook in signal handler */
    if ((signal (SIGALRM, timesignalHandler)) < 0) {
        printf ("ERROR setting signal, errno = %d", errno);
        exit (-1);
    }
    timectl.it_interval.tv_sec = TIMEOUT_IN_SECS;
    timectl.it_interval.tv_usec = TIMEOUT_IN_USECS;
    timectl.it_value.tv_sec = TIMEOUT_IN_SECS;
    timectl.it_value.tv_usec = TIMEOUT_IN_USECS;
    if (setitimer (ITIMER_REAL, &timectl, NULL) < 0) {
        printf ("ERROR setting timer, errno = %d", errno);
        exit (-1);
    }
    /* Setup ICS 110A A/D board */
    setupICS110a();
    /* release 1860(s) */
    1860_kill (AP0pid, SIGUSR1);
    1860_kill (AP1pid, SIGUSR1);
    1860_kill (AP2pid, SIGUSR1);
    1860_kill (AP3pid, SIGUSR1);
    1860_kill (PIAP0pid, SIGUSR1);
    1860_kill (PIAP1pid, SIGUSR1);
    1860_kill (PIAP2pid, SIGUSR1);
    /* map VME devices into virtual space */
    vmeWM = mapVME32 (MM_MEM_ADDRESS, MM_MEM_SIZE);
    if ((int)vmeWM == -1)
        exit (-1);
    /* Host processing */
    #ifdef BOARD_1_SKY
    initNodeValues();
    #endif
    hostShmPIAP0->Status = SYSTEM_START;
    while (hostShmPIAP0->Status != PLATFORM_READY);
    #endif
    #ifdef AA_INSTALLED
    hostShmPIAP1->Status = SYSTEM_START;
    while (hostShmPIAP1->Status != PLATFORM_READY);
    #endif
    #ifdef MOTIF_DISPLAY
    hostShmPIAP2->Status = SYSTEM_START;
    while (hostShmPIAP2->Status != PLATFORM_READY);
    #endif
    #endif
    /* Inform AP0 that host is ready */

```

raccoon/mnt/avds/v5.0/sky/host/hostic

Wed Apr 23 14:30:51 1997

raccoon:/mnt/avds/v5.0/sky/host/host:c

```

hostShmPOAP0->Status[BUFF_A] = SEND_DATA;
hostShmPOAP1->Status[BUFF_A] = PLATFORM_BUSY;
hostShmPOAP2->Status[BUFF_A] = PLATFORM_BUSY;
hostShmPOAP3->Status[BUFF_A] = PLATFORM_BUSY;

#endif

#endif PRINT_HOST_STATUS
if ((cycle & HOST_STATUS_RATE) == 0) {
    printf("Host: status B [0] %d, [1] %d, [2] %d, [3] %d\n",
        hostShmPOAP0->Status[BUFF_B],
        hostShmPOAP1->Status[BUFF_B],
        hostShmPOAP2->Status[BUFF_B],
        hostShmPOAP3->Status[BUFF_B]);
}

#endif

if ((hostShmPOAP1->Status[BUFF_B] == QA_FV_DONE) &&
    (hostShmPOAP2->Status[BUFF_B] == QA_FV_DONE) &&
    (hostShmPOAP3->Status[BUFF_B] == QA_FV_DONE)) {
    /* tell ap0 to send more data to apl, 2 & 3 */
    if (hostShmPOAP0->Status[BUFF_B] == DECODE_DATA) {
        #ifdef MATLAB_DISPLAY
            /* get data frame from micro memory */
            strcpy(dBuff, "getAVDSData");
            sprintf(frameStr, "%d", QAframe);
            strcat(dBuff, frameStr);
            printf("executing %s\n", dBuff);
            system(dBuff);
        /* create a handshake file for matlab */
        system ("touch dataReady; chmod a+rw dataReady");
        #endif
        #ifdef SAVE_OUTPUTS
            /* get outputs from ap's and save */
            mvPOutputs(BUFF_B);
        #endif
        #ifdef MOTIF_DISPLAY
            displayBuff = BUFF_B;
            MinMaxIndex = 0;
        #else
            hostShmPOAP0->Status[BUFF_B] = SEND_DATA;
            hostShmPOAP1->Status[BUFF_B] = PLATFORM_BUSY;
            hostShmPOAP2->Status[BUFF_B] = PLATFORM_BUSY;
            hostShmPOAP3->Status[BUFF_B] = PLATFORM_BUSY;
        #endif
    }

    #ifdef BOARD_1_SKY
        #ifdef MOTIF_DISPLAY
            /* send time series data to display */
            while (MinMaxIndex < NUM_TS_POINTS) {
                sendMinMax(&MinMaxIndex, displayBuff);
            }
            /* send QA data to display */
            if (MinMaxIndex == NUM_TS_POINTS)
                sendQARes(displayBuff);
            MinMaxIndex = 1e32;
        /* send fv data to display */
        if (hostShmPIAP2->Status == SHADE_DONE) {
            printf("Host: SHADING DONE\n");
            dspCmd = FV_PROCESSING;
        }
    #endif A2D
}

SocketWrite(displaySock, &dspCmd, sizeof(int));
SocketWrite(displaySock, &hostShmPIAP2->fvImage[0],
    IMAGE_SIZE*IMAGE_SIZE);
hostShmPIAP2->Status = PLATFORM_READY;

#endif
if ((hostShmPIAP0->Status == RBF_DONE) &&
    (hostShmPIAP1->Status == AA_DONE)) {
    mvPOutputs();
    hostShmPIAP0->Status = PLATFORM_READY;
    hostShmPIAP1->Status = PLATFORM_READY;
}
#endif

#ifdef A2D
    if (hostShmPOAP0->flightStatus == AVDS_DEAD) {
        PLLReset();
        hostShmPOAP0->flightStatus = AVDS_RESET;
        recentReset = 1;
        SocketWrite(PLLSock, "dead", 5);
    }
    if (hostShmPOAP0->flightStatus == AVDS_OK && (recentReset)) {
        recentReset = 0;
        SocketWrite(PLLSock, "good", 5);
    }
}
#endif

#ifdef RUN_FOREVER
    /* Inform AP0 to stop when the proper number of frames is assembled */
    if ((QAframe == numFrames) && (netFrame == numFrames)) {
        printf("Host: Data Acquired, Informing P0/AP0 to stop.\n");
        hostShmPOAP0->Status[0] = AVDS_STOP;
        break;
    }
}
#endif

/* inform P1/AP's to exit */
#ifdef BOARD_1_SKY
    hostShmPIAP0->Status = AVDS_EXIT;
    hostShmPIAP1->Status = AVDS_EXIT;
    #ifdef MOTIF_DISPLAY
        hostShmPIAP2->Status = AVDS_EXIT;
    #endif
    #endif

/* Detach SKY processes and exit */
sleep(2);
retval = boltDetach_all ();

if (retval != 0) {
    printf ("Host: Error on SKY detach, status = %d\n", retval);
    fflush (stdout);
} else {
    printf ("Host: Success on SKY detach\n");
    fflush (stdout);
}

#endif A2D

```

Wed Apr 23 14:30:51 1997

```

/* turn off Phase Lock Loop */
SocketWrite(PLLSock, "stop", 5);
#endif

#ifdef TAPE_INSTALLED
/* tell tape to rewind and stop */
SocketWrite(PLLSock, "tstp", 5);
#endif

/* tell 166 to setup for another run */
SocketWrite(PLLSock, "done", 5);
SocketClose(PLLSock);

#ifdef MOTIF_DISPLAY
dspCmd = AVDS_EXIT;
SocketWrite(displaySock, &dspCmd, sizeof(int));
SocketClose(displaySock);
#endif

#ifdef SAVE_OUTPUTS
close(fdout);
#endif
fclose(frmfile);

printf ("Host: Exiting application \n");
fflush (stdout);

if (ErrorsDuringRun != 0) {
    printf ("\n%&host: %d QA Errors during this acquisition\n\n",
        7, ErrorsDuringRun);
    printf ("Host: Ref %d Total Errors, %d Resets\n",
        tRefErrors, RefResets);
    printf ("Host: Tach %d Total Errors, %d Resets\n",
        tTachErrors, TachResets);
    printf ("Host: MaxChan %d Total Errors, %d Resets\n",
        tMaxChanErrors, MaxChanResets);
    printf ("\n\nHost: Hit any key to continue\n");
    getchar();
}

exit (0);
}

/*****
*
* Function: apSignalCatcher
* OS: SunOS v4.1.1b
* Target: Force 2CE
* Arguments: None
* Returns: N/A
*
* This function services the SIGUSR1 signal from the AP's.
* Rev History: 4/19/95 - (mep) initial design.
*
*****/
void apSignalCatcher()
{
    waitFlag = 0;
    printf("Host: apSignalCatcher: signal processed.\n");
}

/*****
*
* Function: apWaitForSignal
* OS: SunOS v4.1.1b
* Target: Force 2CE
* Arguments: None
* Returns: 1 - signal received, 0 - time out occurred
*
* This function waits for the SIGUSR1 signal from the AP's.
* Rev History: 4/19/95 - (mep) initial design.
*
*****/
int apWaitForSignal ()
{
    int timeout = 10; /* wait for approx. 10 secs. */
    waitFlag = 1;
    while (waitFlag && timeout-- > 0)
        sleep(1);
    printf("Host: apWaitForSignal: wf: %d, to: %d\n", waitFlag, timeout);
    if (timeout == -1)
        return (0);
    else
        return (1);
}

/*****
*
* Function timesignalHandler
*
*****/
void timesignalHandler()
{
    waitForTime = 0;
    cycle++;
}

/*****
*
* Function setupICS110
*
*****/
int setupICS110a()
{
    int stat, i;
}

```

Wed/Apr/23/14:30:51/1997

```

/* sample clock frequency in MHz, 256 times oversampled 16 bit Mode */
/* sample clock frequency in MHz, 128 times oversampled 12 bit Mode */
double actualFreq;

if ((ics110a = ics110a_open("/dev/adco0")) == NULL) {
    printf("setupICS110: Error opening adc0\n");
    return (errno);
}
stat = ics110a_reset(ics110);
if (stat == -1) printf("setupICS110: ERROR ON reset\n");

stat = ics110a_board_reset(ics110);
if (stat == -1) printf("setupICS110: ERROR ON board_reset\n");

#ifdef INTERNAL_CLOCK
stat = ics110a_set_sample_rate(ics110, SAMPLE_RATE, &actualFreq, NUM_BITS);
if (stat == -1) printf("setupICS110: ERROR ON set_sample_rate\n");
else printf("Host: setupICS110: Freq Desired = %f, Actual = %f\n",
    SAMPLE_RATE, actualFreq);
#endif

stat = ics110a_set_output_resolution(ics110, NUM_BITS);
if (stat == -1) printf("setupICS110: ERROR ON set_output_resolution\n");

stat = ics110a_set_acquire(ics110, DISABLED);
if (stat == -1) printf("setupICS110: ERROR ON set_acquire\n");

stat = ics110a_set_master(ics110, DISABLED);
if (stat == -1) printf("setupICS110: ERROR ON set_master\n");

stat = ics110a_set_acquire_source(ics110, EXTERNAL);
if (stat == -1) printf("setupICS110: ERROR ON set_acquire_source\n");

#ifdef ICS_VSB_MODE
stat = ics110a_set_vsb_base_addr(ics110, ICS_VSB_FIFO_UPPER);
if (stat == -1) printf("setupICS110: ERROR ON set_vsb_base_addr\n");

stat = ics110a_set_output_mode(ics110, VSB_D32);
if (stat == -1) printf("setupICS110: ERROR ON set_output_mode\n");
#endif

#ifdef ICS_VME_MODE
stat = ics110a_set_output_mode(ics110, VMEBUS_D32);
if (stat == -1) printf("setupICS110: ERROR ON set_output_mode\n");
endif

stat = ics110a_set_sync_enable(ics110, DISABLED);
if (stat == -1) printf("setupICS110: ERROR ON set_sync_enable\n");

stat = ics110a_set_vme_interrupts(ics110, DISABLED);
if (stat == -1) printf("setupICS110: ERROR ON set_vme_interrupts\n");

stat = ics110a_set_vsb_interrupts(ics110, DISABLED);
if (stat == -1) printf("setupICS110: ERROR ON set_vsb_interrupts\n");

stat = ics110a_set_no_channels(ics110, 10);
if (stat == -1) printf("setupICS110: ERROR ON set_no_channels\n");

stat = ics110a_set_decimation(ics110, 1);
if (stat == -1) printf("setupICS110: ERROR ON set_decimation\n");

```

raccoon/mnt/avds/v5.0/sky/host/host.c

```

)
/*****
/*
/* Function checkICSStatus
/*
/*****
int checkICSStatus()
{
    int stat;
    short sreg;

    stat = ics110a_read_status(ics110, &sreg);
    if (stat == -1) printf("Host: checkICSStatus: ERROR ON read_status\n");
    if (lastICSstat == sreg) {
        lastICSstat = sreg;
        return(1);
    }

    lastICSstat = sreg;

    stat = 1;
    /* [0,1,1,0] - [not started, not full, not 1/2, empty] */
    if (sreg == 0x6) {
        if (QAframe == 0)
            printf("Host: checkICSStatus: waiting for data, sreg=0x%x\n", sreg);
        else
            printf("Host: checkICSStatus: no data, sreg=0x%x\n", sreg);
        stat = 0;
        return(1);
    }
    /* [1,1,1,0] - [started, not full, not 1/2, empty] */
    if (sreg == 0x14) {
        printf("Host: checkICSStatus: dead board, sreg=0x%x\n", sreg);
        stat = 0;
        PLLReset();
        return(1);
    }

    /* got past dead board and no initial data */
    if ((sreg & 0x1) == 0) {
        if (QAframe != 0) {
            printf("Host: checkICSStatus: fifo empty, sreg=0x%x\n", sreg);
            PLLReset();
            stat = 0;
        }
    }
    if ((sreg & 0x4) == 0) {
        if (QAframe != 0) {
            printf("Host: checkICSStatus: fifo full, sreg=0x%x\n", sreg);
            PLLReset();
            stat = 0;
        }
    }
    if ((sreg & 0x8) == 0) {
        printf("Host: checkICSStatus: data acquired, sreg=0x%x\n", sreg);
        stat = 0;
    }
}

```

Wed Apr 23 14:30:51 1997

```

#define PRINT_ICS_STATUS
if (stat)
    printf ("Host: checkICSStatus: sreg=0x%x\n", sreg);
#endif

return(1);
}

/*****
*/
/* Function mvP0Outputs
*/
/*****
*/
int mvP0Outputs(buffer)
int buffer;
{
    int nb;
    int frameStat;
    unsigned long BCB[2];
    unsigned long *vmeAddr;
    unsigned long *answerAddr;
    unsigned char *messageAddr;
    int i;
    int net;
    int node;

    /* calculate ancillary addresses */
    answerAddr = vmeMM + ((VSB_MEM_SIZE - 16) / 4);
    messageAddr = (unsigned char *) answerAddr - (sizeof(struct MESSAGE_IN_REC) + 1);
    zeof(struct MESSAGE_OUT_REC));

    #ifdef A2D
    /* read the BCB */
    BCB[0] = *(vmeMM + ((VSB_MEM_SIZE - 8) / 4));
    BCB[1] = *(vmeMM + ((VSB_MEM_SIZE - 4) / 4));

    /* calc the address where the API QA/FV outputs begin */
    vmeAddr = vmeMM + (BCB[0] / 4) + LONG_BLOCK_SIZE;

    if (buffer == BUFF_A)
        memcpy(vmeAddr, hostShmP0AP1, sizeof(struct ResAccOut));
    else
        memcpy(vmeAddr, hostShmP0AP1->BufBout, sizeof(struct ResAccOut));
    vmeAddr += (sizeof(struct ResAccOut) / 4);

    if (buffer == BUFF_A)
        memcpy(vmeAddr, hostShmP0AP2, sizeof(struct ResAccOut));
    else
        memcpy(vmeAddr, hostShmP0AP2->BufBout, sizeof(struct ResAccOut));
    vmeAddr += (sizeof(struct ResAccOut) / 4);

    if (buffer == BUFF_A)
        memcpy(vmeAddr, hostShmP0AP3, sizeof(struct ResMxOut));
    else
        memcpy(vmeAddr, hostShmP0AP3->BufBout, sizeof(struct ResMxOut));
    #endif BOARD_1_SKY
    /* fill network outputs with -1 if no networks */
    for (node=0; node<NUM_FAULTS; node++)
        for (net=0; net<NUM_RBF_NETS; net++)
            hostShmP1AP0->FaultInd[net][node] = -1;
    for (net=0; net<NUM_AA_NETS; net++)
        hostShmP1AP1->MSEOut[net] = -1;
    /* also save networks outputs with -1 to memory if no networks */
    vmeAddr = vmeMM + (BCB[0] / 4) + LONG_BLOCK_SIZE + LONG_QA_FV_SIZE;
    /* copy the RBF 8x9 outputs */
    memcpy(vmeAddr, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
    vmeAddr += (NUM_RBF_NETS*NUM_FAULTS*4);
    /* copy the average of the 8 net outputs */
    memcpy(vmeAddr, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
    vmeAddr += NUM_FAULTS;
    /* copy the AA output(s) */
    memcpy(vmeAddr, &hostShmP1AP1->MSEOut[0], (NUM_AA_NETS*4));
    vmeAddr += NUM_AA_NETS;
    /* copy the AA average */
    memcpy(vmeAddr, &hostShmP1AP1->MSEOut[0], sizeof(float));
    vmeAddr++;
    /* copy the AA time average */
    memcpy(vmeAddr, &hostShmP1AP1->MSEOut[0], sizeof(float));
    vmeAddr++;
    /* copy the class output */
    i = NoResults;
    memcpy(vmeAddr, &i, sizeof(int));
    /* copy the answer to the immediate answer field */
    memcpy(answerAddr, &i, sizeof(int));
    #ifdef TIMECODE_INSTALLED
    vmeAddr+=3;
    /* should modify to place in report time field */
    memcpy(vmeAddr, &day, sizeof(int));
    vmeAddr++;
    memcpy(vmeAddr, &hour, sizeof(int));
    vmeAddr++;
    memcpy(vmeAddr, &minute, sizeof(int));
    vmeAddr++;
    memcpy(vmeAddr, &second, sizeof(int));
    #endif
    #ifdef SAVE_MESSAGE_DATA
    memcpy(messageBuffer, messageAddr, (sizeof(inMessage)+sizeof(outMessage)));
    vmeAddr++;
    memcpy(vmeAddr, messageBuffer, (sizeof(inMessage)+sizeof(outMessage)));
    #endif

    /* update BCB */
    BCB[0] += SAVE_DATA_SIZE;
    if (BCB[0] == VSB_END_ADDR)
        BCB[0] = INITIAL_VSB_OFFSET;
    if (BCB[0] == BCB[1])
        printf("Host: Data Archiver not keeping up\n");
    *vmeMM + ((VSB_MEM_SIZE - 8) / 4) = BCB[0];
    *vmeMM + ((VSB_MEM_SIZE - 4) / 4) = BCB[1];
    #endif
    #endif

    /* make sure that all the AP's report results for the same frame */

```

racoon/mnt/avds/v5.0/sky/host/host.c

raccoon:/mnt/avds/v5.0/sky/host/host-c

Wed Apr 23 14:30:51 1997

```

nb = write(fout, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
nb = write(fout, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
nb = write(fout, &hostShmP1AP1->MSEout[0], (NUM_AA_NETS*4));
nb = write(fout, &hostShmP1AP1->MSEout[0], sizeof(float));
i = NoResults;
nb = write(fout, &hostShmP1AP1->MSEout[0], sizeof(float));
netFrame++;
#endif /* BOARD_1_SKV */
}
#else
QAframe++;
#endif /* RUN_FOREVER */

/* MOTIF DISPLAY
/* send fv to shading */
if (buffer == BUFF_A) {
    memcpy(&hostShmP1AP2->FVect[0][0],
           &hostShmP0AP1->BufAout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[1][0],
           &hostShmP0AP1->BufAout.AccCh1.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[2][0],
           &hostShmP0AP1->BufAout.AccCh2.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[3][0],
           &hostShmP0AP1->BufAout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[4][0],
           &hostShmP0AP1->BufAout.AccCh1.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[5][0],
           &hostShmP0AP2->BufAout.AccCh2.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[6][0],
           &hostShmP0AP3->BufAout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[7][0],
           &hostShmP0AP3->BufAout.AccCh1.FVect[0],
           sizeof(float)*64);
}
else {
    memcpy(&hostShmP1AP2->FVect[0][0],
           &hostShmP0AP1->BufBout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[1][0],
           &hostShmP0AP1->BufBout.AccCh1.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[2][0],
           &hostShmP0AP1->BufBout.AccCh2.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[3][0],
           &hostShmP0AP2->BufBout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[4][0],
           &hostShmP0AP2->BufBout.AccCh1.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[5][0],
           &hostShmP0AP2->BufBout.AccCh2.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[6][0],
           &hostShmP0AP3->BufBout.AccCh0.FVect[0],
           sizeof(float)*64);
    memcpy(&hostShmP1AP2->FVect[7][0],
           &hostShmP0AP3->BufBout.AccCh1.FVect[0],
           sizeof(float)*64);
}

sizeof(float)*64);
memcpy(&hostShmP1AP2->FVect[6][0],
       &hostShmP0AP3->BufBout.AccCh0.FVect[0],
       sizeof(float)*64);
memcpy(&hostShmP1AP2->FVect[7][0],
       &hostShmP0AP3->BufBout.AccCh1.FVect[0],
       sizeof(float)*64);
}
}
hostShmP1AP2->FrameNum = hostShmP0AP1->FrameNum[buffer];
hostShmP1AP2->Status = DATA_SENT;
#endif

/* send data to nets if no errors */
if ( (ChanErrors <= 1) && (TachErrors == 0) && (RefErrors == 0) ) {
    MaxChanErrors = 0;
    #ifdef BOARD_1_SKV
    if (buffer == BUFF_A) {
        #ifndef RBF_INSTALLED
        memcpy(&hostShmP1AP0->FVect[0][0],
               &hostShmP0AP1->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[1][0],
               &hostShmP0AP1->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[2][0],
               &hostShmP0AP1->BufAout.AccCh2.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[3][0],
               &hostShmP0AP2->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[4][0],
               &hostShmP0AP2->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[5][0],
               &hostShmP0AP2->BufAout.AccCh2.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[6][0],
               &hostShmP0AP3->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP0->FVect[7][0],
               &hostShmP0AP3->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        #endif
        #ifndef AA_INSTALLED
        memcpy(&hostShmP1AP1->FVect[0][0],
               &hostShmP0AP1->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[1][0],
               &hostShmP0AP1->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[2][0],
               &hostShmP0AP1->BufAout.AccCh2.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[3][0],
               &hostShmP0AP2->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[4][0],
               &hostShmP0AP2->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[5][0],
               &hostShmP0AP2->BufAout.AccCh2.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[6][0],
               &hostShmP0AP3->BufAout.AccCh0.FVect[0],
               sizeof(float)*64);
        memcpy(&hostShmP1AP1->FVect[7][0],
               &hostShmP0AP3->BufAout.AccCh1.FVect[0],
               sizeof(float)*64);
        #endif
    }
    #endif
}

```

racon:/mnt/avds/v5.0/sky/host/hostic

Wed Apr 23 14:30:51 1997

```

        &hostShmP0AP2->BufAout.AccCh2.FVect[0],
        sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[6][0],
&hostShmP0AP3->BufAout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[7][0],
&hostShmP0AP3->BufAout.AccCh1.FVect[0],
sizeof(float)*64);
#endif
}
#else
#define RBF_INSTALLED
memcpy(&hostShmP1AP0->FVect[0][0],
&hostShmP0AP1->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[1][0],
&hostShmP0AP1->BufBout.AccCh1.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[2][0],
&hostShmP0AP1->BufBout.AccCh2.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[3][0],
&hostShmP0AP2->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[4][0],
&hostShmP0AP2->BufBout.AccCh1.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[5][0],
&hostShmP0AP2->BufBout.AccCh2.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[6][0],
&hostShmP0AP3->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP0->FVect[7][0],
&hostShmP0AP3->BufBout.AccCh1.FVect[0],
sizeof(float)*64);
#endif
#define AA_INSTALLED
memcpy(&hostShmP1AP1->FVect[0][0],
&hostShmP0AP1->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[1][0],
&hostShmP0AP1->BufBout.AccCh1.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[2][0],
&hostShmP0AP1->BufBout.AccCh2.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[3][0],
&hostShmP0AP2->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[4][0],
&hostShmP0AP2->BufBout.AccCh1.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[5][0],
&hostShmP0AP2->BufBout.AccCh2.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[6][0],
&hostShmP0AP3->BufBout.AccCh0.FVect[0],
sizeof(float)*64);
memcpy(&hostShmP1AP1->FVect[7][0],
&hostShmP0AP3->BufBout.AccCh1.FVect[0],
sizeof(float)*64);

```

```

#endif
        sizeof(float)*64);
}
#define RBF_INSTALLED
        hostShmP1AP0->FrameNum = hostShmP0AP1->FrameNum[buffer];
        hostShmP1AP0->Status = DATA_SENT;
#endif
#define AA_INSTALLED
        hostShmP1AP1->FrameNum = hostShmP0AP1->FrameNum[buffer];
        hostShmP1AP1->Status = DATA_SENT;
#endif
    }
    else {
        #ifdef BOARD_1_SKY
        #ifndef RUN_FOREVER
        #ifdef SAVE_TO_DISK
        printf("mvP0: Writing net data\n");
        /* save to disk file if QA fails */
        nb = write(fcout, &hostShmP1AP0->FaultInd[0][0],
            (NUM_RBF_NETS*NUM_FAULTS*4));
        nb = write(fcout, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
        nb = write(fcout, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
        nb = write(fcout, &hostShmP1AP1->MSEout[0], (NUM_AA_NETS*4));
        nb = write(fcout, &hostShmP1AP1->MSEout[0], sizeof(float));
        i = NoResults;
        nb = write(fcout, &hostShmP1AP1->MSEout[0], sizeof(float));
        netFrame++;
        nb = write(fcout, &i, sizeof(int));
        }
        #endif
        #endif
        /* also save networks outputs with -1 to memory if QA fails */
        /* calc the address where the API QA/FV outputs begin
        vmeAddr = vmem + (BCB[0] / 4) + LONG_BLOCK_SIZE + LONG_QA_FV_SIZE;
        /* copy the RBF 8x9 outputs */
        memcpy(vmeAddr, &hostShmP1AP0->FaultInd[0][0],
            (NUM_RBF_NETS*NUM_FAULTS*4));
        vmeAddr += (NUM_RBF_NETS * NUM_FAULTS);
        /* copy the average of the 8 net outputs */
        memcpy(vmeAddr, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
        vmeAddr += NUM_FAULTS;
        /* copy the RBF_AVERAGE_COUNT final output for fusion */
        memcpy(vmeAddr, &hostShmP1AP0->FaultInd[0][0], (NUM_FAULTS*4));
        vmeAddr += NUM_FAULTS;
        /* copy the AA output(s) */
        memcpy(vmeAddr, &hostShmP1AP1->MSEout[0], (NUM_AA_NETS*4));
        vmeAddr += NUM_AA_NETS;
        /* copy the AA average */
        memcpy(vmeAddr, &hostShmP1AP1->MSEout[0], sizeof(float));
        vmeAddr++;
        /* copy the AA time average */
        memcpy(vmeAddr, &hostShmP1AP1->MSEout[0], sizeof(float));
        vmeAddr++;
        /* copy the class output */
        i = NoResults;
        memcpy(vmeAddr, &i, sizeof(int));
        /* copy the answer to the immediate answer field */
        memcpy(vmeAddr, &i, sizeof(int));
        #ifdef TIMECODE_INSTALLED

```

racoon/mnt/avds/v5.0/sky/host/hostic

Wed Apr 23 14:30:51 1997

```

vmeAddr+=3;
memcpy(vmeAddr, &day, sizeof(int));
vmeAddr++;
memcpy(vmeAddr, &hour, sizeof(int));
vmeAddr++;
memcpy(vmeAddr, &minute, sizeof(int));
vmeAddr++;
memcpy(vmeAddr, &second, sizeof(int));
#endif
#define SAVE_MESSAGE_DATA
memcpy(messageBuffer, messageAddr,
        (sizeof(struct MESSAGE_IN_REC)+sizeof(struct MESSAGE_OUT_REC)));
vmeAddr++;
memcpy(vmeAddr, messageBuffer,
        (sizeof(struct MESSAGE_IN_REC)+sizeof(struct MESSAGE_OUT_REC)));
#endif

/* update BCB */
BCB[0] += SAVE_DATA_SIZE;
if (BCB[0] == VSB_END_ADDR)
    BCB[0] = INITIAL_VSB_OFFSET;
if (BCB[0] == BCB[1])
    printf("Host: Data Archiver not keeping up\n");
*(vmeMM + ((VSB_MEM_SIZE - 8) / 4)) = BCB[0];
*(vmeMM + ((VSB_MEM_SIZE - 4) / 4)) = BCB[1];
#endif
#endif

/* check if MAX_CHAN_ERROR channels are failing */
if (ChanErrors >= MAX_CHAN_ERROR) {
    MaxChanErrors++;
    tMaxChanErrors++;
}

/* reset system if failing QA continuously */
if (MaxChanErrors >= MAX_CHAN_ERROR_RESET_COUNT) {
    MaxChanResets++;
    printf("Host: Resetting PLL on MaxChanErrors\n");
    PLLReset();
    MaxChanErrors = 0;
    TachErrors = 0;
    RefErrors = 0;
    return(0);
}

if (TachErrors >= TACH_ERROR_RESET_COUNT) {
    TachResets++;
    printf("Host: Resetting PLL on TachErrors\n");
    PLLReset();
    MaxChanErrors = 0;
    TachErrors = 0;
    RefErrors = 0;
    return(0);
}

if (RefErrors >= REF_ERROR_RESET_COUNT) {
    RefResets++;
    printf("Host: Resetting PLL on RefErrors\n");
    PLLReset();
    MaxChanErrors = 0;
    TachErrors = 0;
}

RefErrors = 0;
return(0);
}

RefErrors = 0;
return(0);
}

/* Function prtQAError */
void prtQAError(chan, error)
int chan;
int error;
{
    if ((chan < 8) && (error == 4)) {
        #ifdef PRINT_WARNINGS
        printf("prtQAError: QA warning on chan %d\n", chan);
        #else
        return;
        #endif
    }
    else {
        printf("prtQAError: QA error on chan %d\n", chan);
        QAErrors++;
    }
}

#ifdef RING_BELL_ON_ERROR
printf("%c", 7);
#endif

switch (chan) {
    case 8:
        TachErrors++;
        TachErrors++;
        switch (error) {
            case 1:
                printf("prtQAError: Fatal Signal Error\n");
                break;
            case 2:
                printf("prtQAError: Fatal All Channels (Tach Peak Indexing)\n");
                break;
            case 3:
                printf("prtQAError: Fatal All Channels (Tach Peak Count)\n");
                break;
            case 4:
                printf("prtQAError: Fatal All Channels (Tach Pattern Matching)\n");
                break;
        }
        break;
    case 9:
        RefErrors++;
        RefErrors++;
        switch (error) {
            case 1:
                printf("prtQAError: Fatal Signal Level Error\n");
                break;
            case 2:
                printf("prtQAError: Fatal All Channels (MLPE)\n");
                break;
            case 3:
                printf("prtQAError: Fatal All Channels (MDE)\n");
                break;
        }
        break;
}

```

```
racoon:/mnt/avds/v5.0/sky/host/host.c
```

`raccoon:/mnt/avds/v5.0/sky/host/host.c`

Wed Apr 23 14:30:51 1997

```

if (BCB[0] == BCB[1])
    printf("Host: Data Archiver not keeping up\n");
* (vmeMM + ((VSB_MEM_SIZE - 8) / 4)) = BCB[0];
* (vmeMM + ((VSB_MEM_SIZE - 4) / 4)) = BCB[1];
}

/*****
*/
/*      Function RbfAcceptDetermine
*/
/*****
*/
int RbfAcceptDetermine(nodeValues, Fault)
int *nodeValues;
int *Fault;
{
    float maxNodeValue;
    float diff;
    int maxNode;
    int node;
    int i;
    int threshCrossing;

    /* find maximum value for node */
    maxNodeValue = 0.0;
    threshCrossing = 0;
    for (node=0; node < NUM_FAULTS; node++) {
        if (nodeValues[node] > minAcceptValue[node])
            threshCrossing++;
        if (nodeValues[node] > maxNodeValue) {
            maxNodeValue = nodeValues[node];
            maxNode = node;
        }
    }

    #ifdef SINGLE_THRESHOLD
    /* if more than 1 exceeds the threshold, return unacceptable */
    if (threshCrossing > 1)
        return(FALSE);
    #endif

    *Fault = maxNode;
    /* find if difference is acceptable */
    for (node=0; node < NUM_FAULTS; node++) {
        diff = maxNodeValue - nodeValues[node];
        if ((diff < minAcceptDiff[maxNode]) && (node != maxNode))
            break;
    }

    /* determine if this is acceptable */
    if ((maxNodeValue >= minAcceptValue[maxNode]) &&
        (node == NUM_FAULTS))
        return(TRUE);
    else
        return(FALSE);
}

/*****
*/
/*      Function printAnswerText
*/
/*****
*/
void printAnswerText (Answer)
int Answer;
{
    int Report;
    int Fault;
    int Location;
}

```

raccoon:/mnt/avds/v50/sky/host/host.c

Wed Apr 23 14:30:51 1997

[illegible]

racoon:/mnt/avds/v5.0/sky/host/host.c

```

f_ptr6 += 2;
f_ptr7 += 2;
f_ptr8 += 2;
f_ptrTach += 2;
SocketWrite(displaySock, &dspCmd, sizeof(int));
SocketWrite(displaySock, iBuff, sizeof(float)*NUM_TS_PNTS);
delay(40000); /* was 40000 for megatek */
}
*displayIndex += TS_PTS_PER_TICK;
}

/******
/*
/* Function sendQARes
/*
/******
void sendQARes(buffer)
int i;
int *iPtr;
int iBuff[9];

iPtr = iBuff;
if (iPtr == BUFF_A) {
    *iPtr++ = hostShmP0AP1->BufAout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP1->BufAout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP1->BufAout.AccCh2.QARes;
    *iPtr++ = hostShmP0AP2->BufAout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP2->BufAout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP2->BufAout.AccCh2.QARes;
    *iPtr++ = hostShmP0AP3->BufAout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP3->BufAout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP3->BufAout.TachCh.QARes;
}
else {
    *iPtr++ = hostShmP0AP1->BufBout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP1->BufBout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP1->BufBout.AccCh2.QARes;
    *iPtr++ = hostShmP0AP2->BufBout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP2->BufBout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP2->BufBout.AccCh2.QARes;
    *iPtr++ = hostShmP0AP3->BufBout.AccCh0.QARes;
    *iPtr++ = hostShmP0AP3->BufBout.AccCh1.QARes;
    *iPtr++ = hostShmP0AP3->BufBout.TachCh.QARes;
}

hostShmP0AP0->Status[iBuffer] = SEND_DATA;
hostShmP0AP1->Status[iBuffer] = PLATFORM_BUSY;
hostShmP0AP2->Status[iBuffer] = PLATFORM_BUSY;
hostShmP0AP3->Status[iBuffer] = PLATFORM_BUSY;

printf("Buff = %d, QAFRame = %d\n", i, iBuff[i]);
for (i=0; i<9; i++)
    printf("iBuff[%d] = %d\n", i, iBuff[i]);

dspCmd = QA_PROCESSING;
SocketWrite(displaySock, &dspCmd, sizeof(int));
SocketWrite(displaySock, iBuff, sizeof(int)*9);

```

```

}
#endif /* MOTIF_DISPLAY */

```

```

/******
/*
/* Function PLLReset
/*
/******
void PLLReset()
{
    #ifdef A2D
    printf("PLLReset: dropping gate...\n");
    SocketWrite(PLLSock, "stop", 5);
    printf("resetting ICS110A FIFO...\n");
    iCS110a_adc_data_fifo_reset(ics110);
    delay(1000000);
    printf("enabling gate\n");
    SocketWrite(PLLSock, "strt", 5);
    #endif
}

/******
/*
/* Function delay
/*
/******
void delay(count)
int count;
{
    while(count--);
}

```

raccoon:/mnt/avds/v5.0/sky/host/host.c

Wed Apr 23 14:30:53 1997

```

/*****
MODULE:      foxprog.c
AUTHOR:      P. Capes
DATE:        16 MAY 94
DESCRIPTION:  Calculates the 22 bit programming word for a FOX F6151
              programmable oscillator.
ROUTINES:    calcFoxWord()

Copyright 1994 Interactive Circuits and Systems Ltd.
*****/
#include "foxprog.h"

#define getAbsVal(x) ((x)<0)?-(x):(x))

/*****
TITLE:      calcFoxWord
DESC:      Routine to calculate a programming word
PARAM:      freq      desired frequency (MHz)
              progWord returned programming word
RETURN:     OK - driver initialization successful
              ERROR - driver initialization failed
*****/
int calcFoxWord(freq, progWord, progFreq)
double freq;
long *progWord;
double *progFreq;
{
    short m,i,p,pp,q,qp;
    double fvco,fout,err;
    static double freqRange[16]={42.5,47.5,53.5,58.5,62.5,68.5,69.0,84.0,
    87.0,92.0,92.1,105.0,115.0,115.0,115.0,115.0};

    m=0;
    fvco=freq;
    while( (fvco<40.0) && m<8) {
        fvco=fvco*2;
        m++;
    }

    if (m==8) {
        return(ICS150_FREQ_TOO_LOW);
    } else if (fvco>freqRange[15]) {
        return(ICS150_FREQ_TOO_HIGH);
    }

    i=0;
    while ((freqRange[i]<fvco)&&(i<15)) i++;

    p=4;
    q=(int)((double)28.6363*(double)p/(fvco+(double)0.5);
    fout=(28.6363*(double)p/(double)q);
    err=getAbsVal(fvco-fout);
    pp=130-p;
    qp=129-q;
    for (p=5; p<131; p++) {
        q=(int)((double)28.6363*(double)p/(fvco+(double)0.5);

```

```

        fout=(28.6363*(double)p/(double)q);
        if ( (getAbsVal(fvco-fout)<err)&&(q>2)&&(q<72)) {
            err=getAbsVal(fvco-fout);
            pp=p;
            qp=q;
        }
    }

    *progFreq=(28.6363*(double)pp)/((double)qp*(double)(1<<m));
    pp=(130-pp)^0xff;
    qp=(129-qp)^0xff;
    *progWord=((i<0xf)<<18)|((pp<0x7f)<<11)|((0x1<<10)|((m<0x7)<<7)|((qp<0x7f)
    return(OK);
}

```

```

/* H10357 REV.A */
/* ICS110a LIBRARY *****/
#include <stdio.h>
#include <errno.h>
#include </sys/sundev/adcregs.h>
#include </sys/sundev/adccmds.h>

ics110a_open(filename)
char *filename;
/*
 * connects the calling program to the ICS-110a attached to the file
 * named "filename". This routine returns a null pointer if any error
 * occurs.
 */
{
    int fd;
    fd = open(filename, O_RDWR);
    if (fd == -1)
        return(NULL);
    return(fd);
}

ics110a_close(ics110a)
int ics110a;
/*
 * closes the file associated with the ICS-110a pointer ics110a.
 * It returns no errors.
 */
{
    close(ics110a);
    return(0);
}

ics110a_reset(ics110a)
int ics110a;
/*
 * Resets the ICS-110a to its default configuration.
 */
{
    int arg = 0;
    if (ioctl(ics110a, RESET, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_wait_for_vme_interrupt(ics110a)
int ics110a;
/*
 * Puts process to sleep until the ICS-110a interrupts.
 */
{
    int arg = 0;
    if (ioctl(ics110a, WAIT_FOR_INT, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_acquire(ics110a, acquire)
int ics110a;
/*
 * enables or disables acquisition.
 */
{
    int arg = acquire;
    if (ioctl(ics110a, ACQUIRE, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_master(ics110a, master)
int ics110a;
/*
 * master;
 * sets the card as master or slave.
 */
{
    int arg = master;
    if (ioctl(ics110a, MASTER, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_acquire_source(ics110a, source)
int ics110a;
/*
 * source;
 * sets the acquire source.
 */
{
    int arg = source;
    if (ioctl(ics110a, ACQUIRE_SOURCE, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_output_mode(ics110a, mode)
int ics110a;
/*
 * mode;
 * sets the output mode.
 */
{
    int arg = mode;
    if (ioctl(ics110a, OUTPUT_MODE, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_sync_enable(ics110a, sync_enable)
int ics110a;
/*
 * sync_enable;
 * enables or disables the sync word.
 */
{
    int arg = sync_enable;
    if (ioctl(ics110a, SYNC_ENABLE, &arg, 0) != 0)
        return(-1);
}

```

Wed Apr 23 14:31:06 1997

```

return(0);
}

ics110a_set_output_resolution(ics110a,resolution)
int ics110a;
/* resolution;
*
* sets the output resolution.
*/
{
    int arg = resolution;
    if (ioctl(ics110a,OUTPUT_RATE,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_vsb_interrupts(ics110a,vsb_int)
int ics110a;
int vsb_int;
/*
* enables or disables VSBbus interrupts.
*/
{
    int arg = vsb_int;
    if (ioctl(ics110a,VSB_INT_ENABLE,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_vme_interrupts(ics110a,vme_int)
int ics110a;
int vme_int;
/*
* enables or disables VMEbus interrupts.
*/
{
    int arg = vme_int;
    if (ioctl(ics110a,VME_INT_ENABLE,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_no_channels(ics110a,no_channels)
int ics110a;
/* no_channels;
*
* sets the number of channels.
*/
{
    int arg = no_channels-1;
    if (ioctl(ics110a,NO_CHANNELS,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_no_channels_fdp(ics110a,no_channels)
int ics110a;
/* no_channels;
*
* sets the number of channels on the FDP.
*/
{
    int arg = no_channels-1;
    if (ioctl(ics110a,NO_CHANNELS_FDP,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_board_addr_fdp(ics110a,board_address)
int ics110a;
int board_address;
/*
* sets the board address on the FDP.
*/
{
    int arg = board_address;
    if (ioctl(ics110a,BOARD_ADDRESS_FDP,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_decimation(ics110a,decimation)
int ics110a;
int decimation;
/*
* sets the decimation factor.
*/
{
    int arg = decimation-1;
    if (ioctl(ics110a,DECIMATION,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_interrupt_vector(ics110a,vector)
int ics110a;
int vector;
/*
* sets the VMEbus interrupt vector.
*/
{
    int arg = vector;
    if (ioctl(ics110a,INTERRUPT_VECTOR,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_sync_word(ics110a,sync_word)
int ics110a;
int sync_word;
/*
* sets the sync word value.
*/
{
    int arg = sync_word;
    if (ioctl(ics110a,SYNC_WORD,&arg,0) != 0)
        return(-1);
    return(0);
}

ics110a_set_vsb_base_addr(ics110a,base_addr)
int ics110a;

```

racocon:/mnt/avds/v5.0/sky/host/ics110alib.c

Wed Apr 23 14:31:06 1997

```

int
/*
*/
base_addr;
sets the vsb base address.
{
    int arg = base_addr;
    if (!ioctl(ics110a, VSB_BASE_ADDR, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_vsb_space_code(ics110a, space_code)
int
/*
*/
ics110a;
space_code;
sets the vsb space code.
{
    int arg = space_code;
    if (!ioctl(ics110a, VSB_SPACE_CODE, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_block_count_fdp(ics110a, block_count)
int
/*
*/
ics110a;
block_count;
sets the fdp block_count.
{
    int arg = block_count-1;
    if (!ioctl(ics110a, BLOCK_COUNT_FDP, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_set_sample_rate(ics110a, sample_rate, rate, resolution)
int
/*
*/
ics110a;
sample_rate;
resolution;
sets the onboard acquisition clock.
{
    long speed;
    /* a double value representing the desired sampling rate in KHz
    * to calculate the speed of the fox crystal in MHz:
    * for 16 bit mode -> MHz = desired * 0.256
    * for 12 bit mode -> MHz = desired * 0.128
    * call the fox prog sbtrn to calculate an int
    * send this int to the ioctl
    * set the actual speed that the foxprog sbtrn came up with
    */
    /* find out if we are using 16 bit or 12 bit mode */
    if (resolution == BITS16)
        sample_rate = sample_rate * 0.256;
    else
        sample_rate = sample_rate * 0.128;
    calcFoxWord( sample_rate, &speed, (double *)rate);
    printf("Host: ics110a_set_sample_rate: cntlWord: %x\n", speed);
    if (resolution == BITS16)
        *rate = *rate * (1.0/0.256);
    else
        *rate = *rate * (1.0/0.128);
    if (!ioctl(ics110a, PROGRAM_CLOCK, (int)&speed) != 0)
        return(-1);
    return(0);
}

ics110a_set_burst_rate_fdp(ics110a, burst_rate)
int
/*
*/
ics110a;
double burst_rate;
sets the FDP clock.
{
    long speed;
    double actual;
    calcFoxWord( burst_rate, &speed, &actual);
    /* OR the long word created with 0x800000 (set bit 23)
    * this tells the board to set the burst clock, and not
    * the sample clock
    */
    speed = speed | 0x800000L;
    if (!ioctl(ics110a, PROGRAM_CLOCK, (int)&speed) != 0)
        return(-1);
    return(0);
}

ics110a_board_reset(ics110a)
int
/*
*/
ics110a;
resets the ICS-110a.
{
    int arg = 0;
    if (!ioctl(ics110a, BOARD_RESET, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_adc_data_fifo_reset(ics110a)
int
/*
*/
ics110a;
resets the adc data FIFO.

```

raccoon:/mnt/avds/v5.0/sky/host/ics110a/fb.c

Wed Apr 23 14:31:06 1997

```
(
    int    arg = 0;
    if (!ioctl(ics110a, ADC_DATA_FIFO_RESET, &arg, 0) != 0)
        return(-1);
    return(0);
}

ics110a_read_adc_data_fifo(ics110a, buffer, size)
int    ics110a;
long   *buffer;
int    size;
/*
 *   reads "size" data words into "buffer".
 */
{
    if (read(ics110a, buffer, size*2) != size*2)
        return(-1);
    return(0);
}

ics110a_read_status(ics110a, status)
int    ics110a;
short  *status;
/*
 *   reads status register.
 */
{
    if (!ioctl(ics110a, STATUS, status, 0) != 0)
        return(-1);
    return(0);
}
```

raccoon:/mnt/avds/v5.0/sky/host/ics110a/lib.c

Wed Apr 23 14:31:19 1997

```

/*****
 *
 * Program Name:
 *
 * Author: Michael Phillips
 *
 * Module: main routine
 *
 * Function:
 *
 *****/
 *
 * Include Files:
 *
 * -----
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define BLK_SIZE 30720

/*****
 *
 * Global Data
 *
 * -----
 */

char dBuff[BLK_SIZE];
extern errno;

/*****
 *
 * Name: createSockets
 *
 * Purpose: Connect to a named socket as a stream type.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
int createConnectionSocket(servername, port)
char *servername;
int port;
{
    struct sockaddr_in server;
    struct hostent *hp;
    int status, sock;

    status = -1;
    printf("waiting for a connection\n");
    while(status == -1) {
        sock = socket(AF_INET, SOCK_STREAM, 0);
        server.sin_port = port;
        server.sin_family = AF_INET;
        hp = gethostbyname(servername);
        if (hp == 0) {
            fprintf(stderr, "unknown host\n");
            exit(2);
        }
    }
}

```

```

    bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);
    status = connect(sock, (struct sockaddr *)&server, sizeof(server));
    if(status < 0) {
        close(sock);
    }
    printf("connection established...\n");
    return(sock);
}

/*****
 *
 * Name: SocketRead
 *
 * Purpose: reads data sent across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
int SocketRead(sid, source, size)
int sid;
char *source;
int size;
{
    int nbytes;

    nbytes = 0;
    while (nbytes < size) {
        nbytes += read(sid, source, size);
        if (nbytes == -1)
            printf("SocketRead: error reading data, errno = %d\n", errno);
    }
    return (nbytes);
}

/*****
 *
 * Name: SocketWrite
 *
 * Purpose: Sends data across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
SocketWrite(sid, source, size)
int sid;
char *source;
int size;
{
    int nbytes;
    int nbytesTotal;

    nbytesTotal = 0;
    while (nbytesTotal < size) {
        if (size < 1024) {
            nbytes = write(sid, source, size);
            if (nbytes != size) {

```

racocon/mnt/avds/v5.0/sky/host/socket_connect.c

Wed Apr 23 14:31:19 1997

```

if (nbytes == -1)
    printf("SocketWrite: error writing data, errno = %d\n", errno);
else
    printf("SocketWrite: error writing all data %d\n", nbytes);
}
else {
    nbytes = write(sid, source, 1024);
    if (nbytes != 1024) {
        if (nbytes == -1)
            printf("SocketWrite: error writing data, errno = %d\n", errno);
        else
            printf("SocketWrite: error writing all data 0x%x\n", nbytes);
    }
    nbyteTotal += nbytes;
    source += nbytes;
}
}

/***** SocketClose *****/
*
* Name: SocketClose
*
* Purpose: Sends data across a socket connection.
*
* Rev: 02/02/94 (MEP) - Creation.
*
\*****

SocketClose(sid)
int sid;
{
    printf("SocketClose: closing connection\n");
    printf("SocketClose: close status = %d\n", close(sid));
}

/***** sendHandshake *****/
*
* Name: sendHandshake
*
* Purpose: tells server to send next block.
*
* Rev: 02/03/94 (MEP) - Creation.
*
\*****

sendHandshake(sid)
int sid;
{
    int nbytes;
    unsigned short go = 0;

    nbytes = write(sid, &go, sizeof(go));
    if (nbytes != sizeof(go))
        if (nbytes == -1)
            printf("error writing data, errno = %d\n", errno);
        else
            printf("nbytes not equal to size\n");
}

```

raccoon:/mnt/avds/v5.0/sky/host/socket_connect_avds.c

Wed Apr 23 14:37:49 1997

```

/*****
 *
 * File: ap0prog.c
 * Program: ap0prog
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock API
 * Execute: N/A
 * Link list: /usr/sky/bolt/host/lib/skyload.a
 *
 * This program provides the framework for split application with shared
 * memory and interprocessor communication (IPC) using shared memory.
 *
 * Rev History: 16 Mar 95 - dki
 *              19 Apr 95 - mep, modified interface for ipc shared mem
 *              from ap0 -> apl,ap2,ap3. outputs are
 *              now being processed by the host (Sun).
 *
 *****/
\
#define MAIN_PROG
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sky/boltprotos.h>
#include <sky/bolt_ipc.h>
#include <sky/bolt_shm.h>
#include <hostshm.h>
#include <ipcshm.h>

#define CACHE 0x47
#define UCACHE 0x5f /* Sets write through and cache disable */
/*****
 *
 * external declarations
 *
 *****/
extern int bolt_getpid ();
extern char *get_share_mem_addr ();
extern void sleep (unsigned seconds);
extern int apv2vme (unsigned long *);
extern int databst ();
/*****
 *
 * forward declarations
 *
 *****/
void signalCatcher ();
/*****
 *
 * module level variables
 *
 *****/
\

int waitFlag;
/*****
 *
 * Function: main
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock AP0
 * Arguments: None
 * Returns: N/A
 *
 * This function is the "main" routine runs the Platform 0, AP0
 * application.
 *
 * Rev History: 3/15/95 - (dki) initial design.
 *              4/19/95 - (mep) restructured shared memory framework
 *              and setup signal handlers to synchronize
 *              the startup of each AP with the host.
 *
 *****/
\
main( int argc, char *argv[], char *target[] )
{
    int pid;
    int flag = NOFLAGS;

    int shmId10;
    int memkey10 = MEMKEY10;
    unsigned long memsize10 = MEMSIZ10;
    int shmId20;
    int memkey20 = MEMKEY20;
    unsigned long memsize20 = MEMSIZ20;
    int shmId30;
    int memkey30 = MEMKEY30;
    unsigned long memsize30 = MEMSIZ30;
    int shmgetflag = (flag | IPC_CREAT);

    char *shmaddr = 0;
    int shmattrflag = flag;

    int stat; /* return status */

    /* Get pid, split app shared memory address, and make noncache */
    pid = bolt_getpid ();

    printf ("AP0: pid = 0x%x \n", pid);
    fflush (stdout);

    hostShmPOAP0 = (struct IOControl *)get_share_mem_addr();

    printf ("AP0: Split app shared memory address = 0x%x \n", hostShmPOAP0);
    fflush (stdout);

    mprotect ((caddr_t)hostShmPOAP0, sizeof(struct IOControl), UCACHE);

    /* Set split app shared memory buffer pointers */
    hostShmPOAP0->Status[0] = 0;

```

racoon:/mnt/avds/v5.0/sky/platform0/ap0/ap0prog.c

Wed Apr 23 14:31:49 1997

```

/* AP1Shm0 */
shmId10 = bolt_shmget (memkey10, memsize10, shmgetflag);
printf ("AP0: Success on shmget AP1 mem0, shmId10 = %d \n", shmId10);
fflush (stdout);

ipcShmAP1 = (struct BufAccIn *)bolt_shmat (shmId10, shmaddr, shmattrflag);
printf ("AP0: Success on shmat AP1 mem0, ShmAP1 = 0x%x vme = %x\n",
        ipcShmAP1,
        apv2vme((unsigned long *)ipcShmAP1));
fflush (stdout);

/* AP2Shm0 */
shmId20 = bolt_shmget (memkey20, memsize20, shmgetflag);
printf ("AP0: Success on shmget AP2 mem0, shmId20 = %d \n", shmId20);
fflush (stdout);

ipcShmAP2 = (struct BufAccIn *)bolt_shmat (shmId20, shmaddr, shmattrflag);
printf ("AP0: Success on shmat AP2 mem0, ShmAP2 = 0x%x vme = %x\n",
        ipcShmAP2,
        apv2vme((unsigned long *)ipcShmAP2));
fflush (stdout);

/* AP3Shm0 */
shmId30 = bolt_shmget (memkey30, memsize30, shmgetflag);
printf ("AP0: Success on shmget AP3 mem0, shmId30 = %d \n", shmId30);
fflush (stdout);

ipcShmAP3 = (struct BufMixIn *)bolt_shmat (shmId30, shmaddr, shmattrflag);
printf ("AP0: Success on shmat AP3 mem0, ShmAP2 = 0x%x vme = %x\n",
        ipcShmAP3,
        apv2vme((unsigned long *)ipcShmAP3));
fflush (stdout);

/* Setup signal catcher */
i860_signal (SIGUSR1, (void *) signalCatcher);
/* release the host */
host_kill (pid, SIGUSR1);
/* wait for the host to signal */
if (waitForSignal() == 0) {
    printf ("API: timeout occurred waiting for SIGUSR1 signal.\n");
    fflush (stdout);
    exit (1);
}

/* AP0 processing */
/* Begin simple test */
/* Wait for reset release from host */
printf ("AP0: Waiting for initial start from host\n");
fflush (stdout);

while (hostShmP0AP0->Status[0] != SYSTEM_START);

printf ("AP0: Started !! \n");
fflush (stdout);

stat = dataDist (argv[8]);

if (stat != -1) {
    /* allow ap's to finish */
    printf ("AP0: Waiting for AP's to finish !! \n");
    while ((ipcShmAP1->Status[BUFF_A] != PLATFORM_READY) ||
           (ipcShmAP2->Status[BUFF_A] != PLATFORM_READY) ||
           (ipcShmAP3->Status[BUFF_A] != PLATFORM_READY)) {
        while ((ipcShmAP1->Status[BUFF_B] != PLATFORM_READY) ||
               (ipcShmAP2->Status[BUFF_B] != PLATFORM_READY) ||
               (ipcShmAP3->Status[BUFF_B] != PLATFORM_READY)) {
            /* Tell APx to exit */
            ipcShmAP1->Status[BUFF_A] = AVDS_EXIT;
            ipcShmAP2->Status[BUFF_A] = AVDS_EXIT;
            ipcShmAP3->Status[BUFF_A] = AVDS_EXIT;
            ipcShmAP1->Status[BUFF_B] = AVDS_EXIT;
            ipcShmAP2->Status[BUFF_B] = AVDS_EXIT;
            ipcShmAP3->Status[BUFF_B] = AVDS_EXIT;

            printf ("AP0: exiting \n");
            fflush (stdout);
        }
        /* End simple test */
    }
}

/******
 *
 * Function: signalCatcher
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock
 * Arguments: None
 * Returns: N/A
 *
 * This function services the SIGUSR1 signal from the host.
 * Rev History: 4/19/95 - (mep) Initial design.
 *
 ******
 */
void signalCatcher()
{
    waitFlag = 0;
}

```

raccoon:/mnt/avds/v5.0/sky/platform0/ap0/ap0prog.c

Wed Apr 23 14:31:49 1997

```
/* *****\
 *
 * Function: waitForSignal
 *   OS: SKYvec v3.5
 *   Target: SKYbolt Shamrock
 *
 * Arguments: None
 * Returns: 1 - signal received, 0 - time out occurred
 *
 * This function waits for the SIGUSR1 signal from the host
 * Rev History: 4/19/95 - (mep) initial design.
 * *****\
int waitForSignal ()
{
    int timeOut = 100; /* wait for approx. 100 secs. */

    waitFlag = 1;
    while (waitFlag && timeOut--)
        sleep(1);
    if (timeOut == 0)
        return (0);
    else
        return (1);
}
```

raccoon:/mnt/avds/v510/sky/platform0/ap0/ap0prog.c

```

/*****
 *
 * Program: dataDist
 * OS: SKYvect v3.5
 * Target: Skybolt Shamrock
 * Author: M. E. Phillips, NRAD, Code 535
 * email: philipm@nosc.mil
 *
 * This program acquires data from the ICS 110 analog input board,
 * demultiplexes it into the AVDS frame format, and distributes the
 * demultiplexed samples to both an AP and memory storage. Design uses
 * scalar routines to perform a two-stage demux that is optimized for the
 * SKYbolt architecture.
 *
 * Rev History: 4/12/95 (MEP) - initial design. Runs with continuous
 * data sampling.
 * 4/13/95 (MEP) - modified to incorporate data stream
 * from a file or A/D
 * 5/11/95 (MEP) - modified to retrieve data over VSB
 * to enable data collection from the
 * ICS-110A board. Also added packet
 * switching (DEMUX cycle A and TRANSFER
 * cycle B)
 *
 *****/
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <vector.h>
#include <system.h>
#include <mathlib.h>
#include <sky/boltprotos.h>
#include <sky/burst_services.h>
#include <sky/dma_services.h>
#include <hostshm.h>
#include <ipcshm.h>
#include <dataDist.h>

#define RUN_FOREVER
#define A2D
#define RUN_AVDS
#define A2D
#define PLAYBACK
#define MEM_IO
#define FILEMODE
#define FILE_IO
#define

/* included facilities */

/*****
 *
 * Send timeseries data to VSB memory */
#define VSB_OUTPUT 2
/* Send timeseries data to AP123 */
#define IBUS_OUTPUT 1

```

```

#define DEMUX_DATA 0 /* Demux timeseries data */
#define ICS_VSB_MODE /* Take samples from ICS110A VSB FIFO */
/*****
 *
 * excluded facilities
 *
 *****/
\
#define MOTIF_DISPLAY /* calculate time series values for display */
#define STOP_ON_ERROR /* exit processing if errors occur */
#define ICS_FIFO_TEST_MODE /* Test FIFO integrity (test mode only) */
#define ICS_VME_MODE /* Take samples from ICS110A VME FIFO */
#define PRINT_ALL /* Display debug on demux */
#define LOOPING_OUTPUT /* Direct writes over IBUS (non-DMA) */
#define DEBUG /* Display debug information */
#endif

/*****
 *
 * external declarations
 *
 *****/
extern void sleep( unsigned );
extern void usleep( unsigned );
extern int bolt_getpid ( );
extern void *memcpy ( void *, void *, int );
extern int apv2sp( UL * );
extern int apv2vme( int * );

/* forward declarations */

/*****
 *
 * void printSetup();
 * int demuxBuffer(UL *, UL *);
 * int demuxSamplesFast(short *, UL *);
 * int demuxSamplesVect(UL *, UL *);
 * int fillSkyBuff(UL *);
 * int close_stream( int );
 * int write_data( int, UL * );
 * int open_stream( char * );
 *
 *****/
/* local constants */
\
/*****
 *
 * addresses for the ICS 110 FIFO, status & control registers */
#define ICS_STATUS_ADDR 0xb8008000
#define ICS_CONTROL_ADDR 0xb8000000
#define DEAD_COUNT_56SEC 250000
#define NODATA_START_TIMEOUT (DEAD_COUNT_56SEC / 2)
#define NODATA_NORM_TIMEOUT (DEAD_COUNT_56SEC / 4)

```

tacocon/mnt/avds/v5.0/sky/platform0/ap0/dataDist.c

```

/* offset constants for a/b buffers */
#define IBUS_BUFF_A 0
#define IBUS_BUFF_B12 (sizeof(struct DataAccIn) / 4)
#define IBUS_BUFF_B3 (sizeof(struct DataMixIn) / 4)
/*****
 *
 * Function: dataDist
 * OS: SKYvect v3.5
 * Target: Skybolt Shamrock
 * Author: M. E. Phillips, NRAd, Code 535
 * Arguments: None
 * Returns: N/A
 *
 * This function is the routine that provides the processing loop
 * for gathering the A/D data from the ics110, demultiplexing it, and
 * distributing it to all destinations.
 *
 * Rev History: 4/11/95 - initial design-ep)
 * 5/14/95 - (mep) modified to use vsb bus instead of
 * front panel for ICS110A support. Design
 * sends data in alternating cycles, i.e.
 * demux -> Transfer.
 *
 *****/
int dataDist ( filename )
char *filename;
(
    int    buffer,
    frameCount = 0,
    bolt_pid,
    /* loop counter for buffer transfers
    /* number of buffer transfers
    /* SKY process id
    bufferFlag = VSB_OUTPUT, /* flag for data control
    busXfrSize = 0,
    /* size for variable transfer size
    busOffset,
    /* offset to current ibus xfr block
    busDest = 0,
    /* Ibus dest address
    sample = 0,
    /* counter for file io decoding
    bufIndex,
    /* index into a,b status and framecount
    deadCount,
    /* watchdog counter for ICS FIFO

    UL
    *skyBuff,
    /* skyBurst data is read to here
    *passBuff,
    /* 1K x 10 buffer
    *demuxBuff,
    /* 64K x 10 buffer
    *a2dBuff,
    /* A/D dest buffer address
    *outBuff,
    /* buffer to "page" to final VSB dest
    *busDest,
    /* VSB dest address
    *1860Addr = 0,
    /* place holder for ibusDest
    *fileLong = 0;

    UL    BCB[2];
    /* offset to next buffer to write to [0]
    /* or read from [1]

    short *ics_status,
    *ics_control;

    ifndef A2D
    int  iread;
    endif
    /* number of values read

```

```

#define FILE_IO
int  fd = 0;
short *fileShort = 0,
/* file descriptor for file input mode
/* short pointer to data buffer
/* placeholder for data channel swapping
/* counter for data channel swapping
*/
int  chanSet;
#endif

#define MEM_IO
int  mem_status = 0;
int *stat = &mem_status;
/* status of data input system
/* pointer to mem_status
*/
#endif

printSetup();

bolt_pid = bolt_getpid ( );

/* allocate memory buffers
ics_status = (short *) valloc ( sizeof( short ) );
ics_control = (short *) valloc ( sizeof( short ) );
skyBuff = (UL *) valloc ( BUFFER_SIZE );
if (skyBuff == NULL) {
    printf("dataDist: ERROR: Can not allocate storage for SkyBurst buffer\n");
    return( -1 );
}
passBuff = (UL *) valloc ( BUFFER_SIZE );
if (passBuff == NULL) {
    printf("dataDist: ERROR: Can not allocate storage for 1K x 10 buffer\n");
    return( -1 );
}
demuxBuff = (UL *) valloc ( BLOCK_SIZE );
if (demuxBuff == NULL) {
    printf("dataDist: ERROR: Can not allocate storage for demux buffer\n");
    return( -1 );
}

/* place some data in UNCACHE pool for I/O purposes.
/* NOTE: placing the I/O buffers into the UNCACHE pool will decrease
/* performance. Suffering the cache flushes prior to I/O causes
/* 10 to 1 less overhead then processing out of the uncache pool
mprotect ( (caddr_t)ics_status, sizeof( short ), UNCACHE );
mprotect ( (caddr_t)ics_control, sizeof(long)*2, UNCACHE );
mprotect ( (caddr_t)BCB, sizeof(long)*2, UNCACHE );
mprotect ( (caddr_t)ipcShmAP1, sizeof(ipcShmAP1), UNCACHE );
mprotect ( (caddr_t)ipcShmAP2, sizeof(ipcShmAP2), UNCACHE );
mprotect ( (caddr_t)ipcShmAP3, sizeof(ipcShmAP3), UNCACHE );

/* fill dummy buffer, if not using the A/D as data source
*/
ifndef A2D
fillSkyBuff (skyBuff);
#endif

/* support input from a file, format is same as A/D stream
*/
ifndef FILE_IO
printf("dataDist: filename is %s\n", filename);
fd = openStream(filename);
if (fd == 0) {
    printf("ERROR opening file %s\n", filename);
    return(-1);
}

```

Wed Apr 23 14:31:53 1997

```

)
#endif

printf ( "dataDist: bufIndex address (VME): 0x%x\n", apv2vme(&bufIndex));
printf ( "dataDist: buffFlag address (VME): 0x%x\n", apv2vme(&buffFlag));
printf ( "dataDist: Starting Data Collection.\n" );

/* Inform host that data distribution is ready */
hostShmPOAP0->Status[0] = PLATFORM_READY;

#ifdef MEM_IO
/* wait for data to arrive */
deadCount = 0;
do {
    /* get BCB from memory board */
    move_to_BOLT((UL)VSB_MEM_WRITE_POINTER_ADDR, 1, (UL)BCB, 2,
        VSB_BUS | TRANS_WALL | LONG_DWA | SINGLE_DWA | VSB_SYS | BLKS512);
    sleep(1);
    deadCount++;
    if ((deadCount%10) == 0)
        printf ("dataDist: Waiting for Tape Drive.\n");
} while (BCB[1] != 0);
#else
/* initialize BCB */
BCB[0] = 0;
BCB[1] = 0;
move_from_BOLT((UL)BCB, (UL)VSB_MEM_WRITE_POINTER_ADDR, 1, 2,
    VSB_BUS | TRANS_WALL | LONG_DWA | SINGLE_DWA | VSB_SYS | BLKS512);
#endif

/* start infinite processing loop */
while (1) {
    /* calc VSB destination or source for MEM_IO mode */
    #ifdef MEM_IO
    /* wait for data to arrive */
    deadCount = 0;
    do {
        /* get BCB from memory board */
        move_to_BOLT((UL)VSB_MEM_WRITE_POINTER_ADDR, 1, (UL)BCB, 2,
            VSB_BUS | TRANS_WALL | LONG_DWA | SINGLE_DWA | VSB_SYS | BLKS512);
        sleep(1);
        deadCount++;
        if ((deadCount%10) == 0) {
            /* get STATUS from memory board */
            move_to_BOLT((UL)VSB_MEM_STATUS_POINTER_ADDR, 1, (UL)stat, 1,
                VSB_BUS | TRANS_WALL | LONG_DWA | SINGLE_DWA | VSB_SYS | BLKS512);
            if (mem_status == -1) {
                printf("dataDist: Exiting on Tape Error or EOF.\n");
                return (-1);
            }
            printf ("dataDist: Waiting for Data, stat = %d.\n");
        } while (BCB[0] == BCB[1]);
        VSBDest = (unsigned long *) (VSB_MEM_BASE_ADDR + BCB[1]);
        #else
        VSBDest = (unsigned long *) (VSB_MEM_BASE_ADDR + BCB[0]);
        #endif
    }

    #ifdef DEBUG
    move_to_BOLT( (UL)ICS_STATUS_ADDR, 0, (UL)ics_status, 1,

```

racocon:/mnt/avds/v5.0/sky/platform0/ap0/dataDist.c

Wed Apr 23 14:31:53 1997

```

VSB_BUS | TRANS_WALL | LONG_DMA | SINGLE_DMA | VSB_SYS );
) while ((*ics_status) & 0x2000);
move_to_BOLT((UL)ICS_VSB_FIFO, 0, (UL)skyBuff, LONG_BUFFER_SIZE/2,
VSB_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | VSB_SYS | BLKS2k);
move_to_BOLT((UL)ICS_VSB_FIFO, 0, (UL)skyBuff, LONG_BUFFER_SIZE/2,
VSB_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | VSB_SYS | BLKS2k);
)
#endif
#define ICS_VSB_MODE
/* Wait for FIFO 1/2 full on ICS.
deadCount = 0;
do {
    move_to_BOLT((UL)ICS_STATUS_ADDR, 0, (UL)ics_status, 1,
VSB_BUS | TRANS_WALL | LONG_DMA | SINGLE_DMA | VSB_SYS );
    deadCount++;
    if (((deadCount >= NODATA_NORM_TIMEOUT) && (frameCount != 0)) ||
        (deadCount >= NODATA_START_TIMEOUT)) {
        printf("dataDist: ICS110A A/D board not running\n");
        hostShmPOAPO->flightStatus = AVDS_DEAD;
        deadCount = 0;
        usleep(100);
    }
} while ((*ics_status) & 0x2000);
move_to_BOLT((UL)ICS_VSB_FIFO, 0, (UL)skyBuff, LONG_BUFFER_SIZE/2,
VSB_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | VSB_SYS | BLKS1k);
move_to_BOLT((UL)ICS_VSB_FIFO, 0, (UL)(skyBuff+(LONG_BUFFER_SIZE/2)),
LONG_BUFFER_SIZE/2,
VSB_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | VSB_SYS | BLKS1k);
hostShmPOAPO->flightStatus = AVDS_OK;
#endif
#define ICS_VME_MODE
do {
    move_to_BOLT((UL)ICS_STATUS_ADDR, 0, (UL)ics_status, 1,
VME_BUS | TRANS_WALL | SHORT_DMA | SINGLE_DMA | A24 );
} while ((*ics_status) & 0x0002);
move_to_BOLT((UL)ICS_VME_FIFO, 1, (UL)skyBuff, LONG_BUFFER_SIZE/2,
VME_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | A24);
/* Check for FIFO 1/2 full on ICS.
do {
    move_to_BOLT((UL)ICS_STATUS_ADDR, 0, (UL)ics_status, 1,
VME_BUS | TRANS_WALL | SHORT_DMA | SINGLE_DMA | A24 );
} while ((*ics_status) & 0x0002);
move_to_BOLT((UL)ICS_VME_FIFO, 1, (UL)(skyBuff+(LONG_BUFFER_SIZE/2)),
LONG_BUFFER_SIZE/2,
VME_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | A24);
#endif
#define A2D
ifread = BUFFER_SIZE;
usleep(5000);
#endif
#define FILE_IO
if (buffFlag == DEMUX_DATA) {
    ifread = read(fd, (char *)skyBuff, BUFFER_SIZE);
    if (ifread != BUFFER_SIZE) {
        printf("dataDist: END OF FILE or ERROR reading data file\n");
        closeStream(fd);
        return (-1);
    }
}

fileShort = (short *)skyBuff;
chanSet = SHORT_SAMPLES_PER_BUFFER;
while (chanSet--) {
    tempShort = *fileShort++; /* save 100Hz into temp */
    sample = NUM_CHAN - 1;
    while (sample--)
        *(fileShort - 1) = *fileShort++;
    fileShort--;
    *fileShort++ = tempShort;
}
#endif
#define MEM_IO
if (buffFlag == DEMUX_DATA)
    move_to_BOLT((UL)VSBDEST, 0, (UL)skyBuff, LONG_BUFFER_SIZE,
VSB_BUS | TRANS_WALL | LONG_DMA | BLOCK_DMA | VSB_SYS | BLKS1k);
#endif
#define DEBUG
printf("dataDist: After burst_read - buffer %d - ", buffer);
printf("dataDist: Got 0x%x bytes.\n", ifread);
fflush(stdout);
#endif

/* only process alternating cycles
switch (buffFlag) {
case DEMUX_DATA:
    /* demultiplex
    #ifdef MEM_IO
    demuxSamplesFast((short *)skyBuff, passBuff);
    #else
    demuxBuffer(passBuff, a2dBuff);
    #endif
    memcpy(a2dBuff, skyBuff, 0x5000);
    #endif
    break;
case IBUS_OUTPUT:
    /* write to Ap's
    /* If switching Ap's reset the offset to 0
    if ((buffer == 47) || (buffer == 31))
        ibusOffset = 0;
    /* select which Ap based on the decrementing index "buffer"
    if (buffer > 47) {
        i860Addr = ((unsigned long *)ipcShmAP1) + ibusOffset;
        if (buffIndex == 0)
            i860Addr += IBUS_BUFF_A;
        else
            i860Addr += IBUS_BUFF_B12;
        ibusXfrSize = LONG_24K;
    }
    else if (buffer > 31) {
        i860Addr = ((unsigned long *)ipcShmAP2) + ibusOffset;
        if (buffIndex == 0)
            i860Addr += IBUS_BUFF_A;
        else
            i860Addr += IBUS_BUFF_B12;
        ibusXfrSize = LONG_24K;
    }
    else if (buffer > 10) {
        i860Addr = ((unsigned long *)ipcShmAP3) + ibusOffset;
        if (buffIndex == 0)
            i860Addr += IBUS_BUFF_A;
        else
            i860Addr += IBUS_BUFF_B12;
    }
}

```

Wed Apr 23 14:31:53 1997

```

1860Addr += IBUS_BUFF_B3;
ibusXfrSize = LONG_24K;
}
else if (buffer == 10) {
1860Addr = (((unsigned long *)ipcShmAP3) + ibusOffset);
if (bufIndex == 0)
1860Addr += IBUS_BUFF_A;
else
1860Addr += IBUS_BUFF_B3;
ibusXfrSize = LONG_8K;
}
/* transfer only the 1st 54 blocks (53@24K + 1@8K = 1280K) */
if (buffer > 9) {
#ifdef DEBUG
printf ("dataDist: writing to 1860 address %x, offset %x, size %x\n",
1860Addr,
ibusOffset,
ibusXfrSize);
#endif
#ifdef IBUS_OUTPUT
ibusDest = apv2sp(1860Addr);
move_from_BOLT((UL)outBuff2, (UL)ibusDest, 1, ibusXfrSize,
move_IDMA_BUS | TRANS_NOWAIT | LONG_DMA | SINGLE_DMA);
#endif
}
break;
case VSB_OUTPUT:
#ifdef DEBUG
printf ("dataDist: BCB_Write = %x, BCB_Read = %x\n", BCB[0], BCB[1]);
#endif
#ifdef MEM_IO
/* if MEM_IO data is received from VSB memory and not written. */
/* so the read pointer is updated for MEM_IO and the write for */
/* all other modes.
BCB[1] += SAVE_DATA_SIZE;
if (BCB[1] == VSB_END_ADDR)
BCB[1] = INITIAL_VSB_OFFSET;
move_from_BOLT((UL)BCB, (UL)VSB_MEM_WRITE_POINTER_ADDR, 1, 2,
VSB_BUS | TRANS_WALL | LONG_DMA | SINGLE_DMA | VSB_SYS | BLKS512);
*/
/* update the offset of where to write the next data block */
/* NOTE: Host now updates the BCB to allow the host to update */
/* the data blocks with intermediate results
BCB[0] += SAVE_DATA_SIZE;
if (BCB[0] == VSB_END_ADDR)
BCB[0] = INITIAL_VSB_OFFSET;
#endif
frameCount++; /* increment processed frame counter */
break;
}
/* exit processing loop when signaled from the host */
if (hostShmPOAP0->Status[0] == AVDS_STOP) {
printf ("dataDist: Received Stop from host.\n");
break;
}
} /* end while(1) */
}
/* Turn off ICS.
move_to_BOLT((UL)ICS_CONTROL_ADDR, 0, (UL)ics_control, 1,
VME_BUS | TRANS_WALL | SHORT_DMA | SINGLE_DMA | A24 );
*ics_control &= 0;
move_from_BOLT((UL)ics_control, (UL)ICS_CONTROL_ADDR, 0, 1,
VME_BUS | TRANS_WALL | SHORT_DMA | SINGLE_DMA | A24 );
*/
}
/* Check for FIFO full on ICS.
move_to_BOLT((UL)ICS_STATUS_ADDR, 0, (UL)ics_status, 1,
VSB_BUS | TRANS_WALL | LONG_DMA | SINGLE_DMA | VSB_SYS );
if (((*ics_status) & 0x4000) == 0) {
fprintf(stderr, "dataDist: FIFO FULL, ics stat = %x\n", *ics_status);
/* Turn off ICS.

```

racocon:/mnt/avds/v5.0/sky/platform0/ap0/dataDist.c

```

printf ( "dataDist: Terminating Data Collection.\n" );
/* close all open devices
#ifdef FILE_IO
closeStream(fd);
#endif
*/
printf ( "dataDist: About to return, %d data frames processed\n",
return ( 0 );
)

/*****
*
* Function: printSetup
* OS: SKYvect v3.5
* Target: Skybolt Shamrock
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: None
* Returns: N/A
*
* This function prints out the defines to the display.
*
* Rev History: 4/11/95 - Initial design.
*
*****/
void printSetup()
{
    printf ("\n" );
    printf ("dataDist: AVDS Data Distributor Rev 4.0 *****\n\n" );
    printf ("dataDist: ICS_STATUS_ADDR = 0x%x\n", ICS_STATUS_ADDR);
    printf ("dataDist: ICS_CONTROL_ADDR = 0x%x\n", ICS_CONTROL_ADDR);
    printf ("dataDist: ICS_VSB_FIFO = 0x%x\n", ICS_VSB_FIFO);
    printf ("dataDist: VSB_MEM_BASE_ADDR = 0x%x\n", VSB_MEM_BASE_ADDR);
    printf ("dataDist: VSB_MEM_SIZE = 0x%x\n", VSB_MEM_SIZE);
    printf ("dataDist: INITIAL_VSB_OFFSET = 0x%x\n", INITIAL_VSB_OFFSET);
    printf ("dataDist: VSB_END_ADDR = 0x%x\n", VSB_END_ADDR);
    printf ("dataDist: WRITE_POINTER_ADDR = 0x%x\n", VSB_MEM_WRITE_POINTER_ADDR);
    printf ("dataDist: READ_POINTER_ADDR = 0x%x\n", VSB_MEM_READ_POINTER_ADDR);
    printf ("dataDist: SAVE_DATA_SIZE = 0x%x\n", SAVE_DATA_SIZE);
    printf ("dataDist: BLOCK_SIZE = 0x%x\n", BLOCK_SIZE);
    printf ("dataDist: NUM_CHAN = 0x%x\n", NUM_CHAN);
    printf ("dataDist: SAMPLES_PER_CHANNEL = 0x%x\n", SAMPLES_PER_CHANNEL);
    printf ("dataDist: LONG_SAMPLES_PER_CHANNEL = 0x%x\n", LONG_SAMPLES_PER_CHANNEL);
    printf ("dataDist: BUFFER_SIZE = 0x%x\n", BUFFER_SIZE);
    printf ("dataDist: SHORT_SAMPLES_PER_BUFFER = 0x%x\n", SHORT_SAMPLES_PER_BUFFER);
    printf ("dataDist: LONG_SAMPLES_PER_BUFFER = 0x%x\n", LONG_SAMPLES_PER_BUFFER);
    printf ("dataDist: LONG_BUFFER_SIZE = 0x%x\n", LONG_BUFFER_SIZE);
    printf ("dataDist: BUFFERS_PER_BLOCK = 0x%x\n", BUFFERS_PER_BLOCK);
    printf ("dataDist: NUM_BLOCKS = 0x%x\n", NUM_BLOCKS);
    printf ("\n");
}

/*****
*
* Function: demuxBuffer
* OS: SKYvect v3.5
* Target: Skybolt Shamrock
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: src - pointer to the source buffer
*            dest - pointer to the destination buffer
* Returns: status
*
* This function is the scalar version of the (1024 x 10) -> (64k x 10)
* demultiplexing routine (Stage 2).
*
* Rev History: 4/11/95 - Initial design.
*
*****/
int demuxBuffer(UL *src, UL *dest)
{
    int chan;

    chan = NUM_CHAN;
    while (chan--) {
#ifdef PRINT_ALL
        printf ("demuxBuffer: src = %x, dest = %x\n", src, dest);
#endif
        memcpy(dest, src, BYTE_SAMPLES_PER_BUFFER);
        src += LONG_SAMPLES_PER_BUFFER;
        dest += LONG_SAMPLES_PER_CHANNEL;
    }
    return 1;
}

/*****
*
* Function: demuxSamplesFast
* OS: SKYvect v3.5
* Target: Skybolt Shamrock
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: src - pointer to the source buffer
*            dest - pointer to the destination buffer
* Returns: status
*
* This function is the scalar version of the (1 x 10 x 1024) -> (1024 x 10)
* demultiplexing routine (Stage 1).
*
* Rev History: 4/11/95 - Initial design.
*
*****/
int demuxSamplesFast(short *src, UL *dest)
{
    UL *chanDest;
    int chan;
    sample;

    sample = LONG_SAMPLES_PER_BUFFER;
    while(sample--) {

```

Wed Apr 23 14:31:53 1997

```

chanDest = dest;
chan = NUM_CHAN;
while(chan--) {
    #ifdef PRINT_ALL
    #ifdef DEBUG
    printf ("demuxSamplesFast: src = %.8x, chanDest = %.8x, dest = %.8x\n",
           src, chanDest,
           dest);
    sleep(1);
    #endif
    *((short *)chanDest + 1) = *(src + NUM_CHAN);
    *(short *)chanDest = *src++;
    chanDest += LONG_SAMPLES_PER_BUFFER;
}
src += NUM_CHAN;
dest++;
}
return 1;
}

/*****
 *
 * Function: fillSkyBuff
 * OS: SKYvect v3.5
 * Target: Skybolt Shamrock
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: buff - pointer to array to fill
 * Returns: status
 *
 * This function fills the buffer with a test pattern:
 *
 *   chan 0,1 - 00010002
 *   chan 2,3 - 00030004
 *   chan 4,5 - 00050006
 *   chan 6,7 - 00070008
 *   chan 8,9 - 0009000a
 *   ...
 *
 * This emulates the A/D board output where data (16bits) is in bits 31-16 and
 * a sync bit is in bit 1 of each data word. The multiplexed data repeats this
 * format continuously.
 *
 * Rev History: 4/11/95 - initial design.
 *****/
int fillSkyBuff(UL *buff)
{
    int sample,
        chan;

    for (sample=0; sample<SHORT_SAMPLES_PER_BUFFER; sample++)
        for (chan=0; chan<NUM_CHAN/2; chan++) {
            *((short *)buff + chan*2+1) =
                *((short *)buff + 1) = chan*2+2;
            #ifdef PRINT_ALL
            printf("samp=%d, chan=%d, value=%.8x\n", sample, chan, *buff);
            #endif
        }
}

chanDest = dest;
chan = NUM_CHAN;
while(chan--) {
    #ifdef PRINT_ALL
    #ifdef DEBUG
    printf ("demuxSamplesFast: src = %.8x, chanDest = %.8x, dest = %.8x\n",
           src, chanDest,
           dest);
    sleep(1);
    #endif
    *((short *)chanDest + 1) = *(src + NUM_CHAN);
    *(short *)chanDest = *src++;
    chanDest += LONG_SAMPLES_PER_BUFFER;
}
src += NUM_CHAN;
dest++;
}
return 1;
}

/*****
 *
 * Function: closeStream
 * OS: SKYvect v3.5
 * Target: Skybolt Shamrock
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: filedesc - file descriptor
 * Returns: status
 *
 * This function closes the file associated with filedesc.
 *
 * Rev History: 4/11/95 - initial design.
 *****/
int closeStream( int filedesc )
{
    if (close (filedesc) < 0)
    {
        fprintf (stderr, "Unable to close stream %d\n", filedesc);
        return( 0 );
    }
    return 1;
}

/*****
 *
 * Function: openStream
 * OS: SKYvect v3.5
 * Target: Skybolt Shamrock
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: filedesc - file descriptor
 * Returns: status
 *
 * This function opens the file associated with filedesc.
 *
 * Rev History: 4/11/95 - initial design.
 *****/
int openStream( char *filespec )
{
    int filedesc;

    filedesc = open( filespec, O_RDONLY );
    if (filedesc < 0)
    {
        fprintf(stderr, "Output stream [%s] open failed\n", filespec);
        return( 0 );
    }
    return filedesc;
}

/*****
 *
 * Function: openStreamWR
 *****/

```

racocon/mnt/avds/v5.0/sky/platform0/ap0/dataDist.c

Wed Apr/23/14:31:53 1997

```
* OS: SKYvect v3.5
* Target: Skybolt Shamrock
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: filedesc - file descriptor
* Returns: status
*
* This function opens the file associated with filedesc.
*
* Rev History: 4/11/95 - initial design.
*\*****
int openStreamWR( char *filespec )
{
    int filedesc;

    filedesc = open( filespec, O_RDWR | O_CREAT, 0644 );
    if (filedesc < 0)
    {
        fprintf(stderr, "Output stream [%s] open failed\n", filespec);
        return( 0 );
    }
    return filedesc;
}
```

racoon:/mnt/avds/v5.0/sky/platform0/ap0/dataDist.c

Wed Apr 23 14:32:20 1997

```

/*****
 *
 * File: Plap0prog.c
 * Program: Plap0prog
 * OS: SKIvec V3.5
 * Execute: N/A
 * Link list:
 *
 * This program provides the framework for split application with shared
 * memory and interprocessor communication (IPC) using shared memory.
 *
 * Rev History: 16 Mar 95 - dkl
 *              19 Apr 95 - mep, modified interface for ipc shared mem
 *              from ap0 -> apl,ap2,ap3. outputs are
 *              now being processed by the host (Sun).
 *
 *****/
/*****
 *
 * define MAIN_PROG
 *
 * include <stdio.h>
 * include <stdlib.h>
 * include <errno.h>
 * include <signal.h>
 * include <sys/types.h>
 * include <sky/boltprotos.h>
 * include <sky/bolt_ipc.h>
 * include <sky/bolt_shm.h>
 * include <memory.h>
 * include "hostshm.h"
 *
 * Define CACHE 0x47
 * Define UCACHE 0x5f /* Sets write through and cache disable */
 *
 *****/
/*****
 *
 * external declarations
 *
 *****/
int rbf_recog();
int bolt_getpid();
char *get_share_mem_addr();
void usleep (unsigned seconds);
void sleep (unsigned seconds);
int apv2vme(void *);

/*****
 *
 * forward declarations
 *
 *****/
void signalCatcher();
void getFV();

/*****
 *
 * module level variables
 *
 *****/

```

```

int waitFlag;
float FVarray[64*8*10];
/*****
 *
 * Function: main
 * OS: SKIvec v3.5
 * Target: SKYbolt Shamrock API
 * Arguments: None
 * Returns: N/A
 *
 * This function is the "main" routine runs the Platform 1, APO
 * application.
 *
 * Rev History: 3/15/95 - (dkl) initial design.
 *              4/19/95 - (mep) restructured shared memory framework
 *              and setup signal handlers to synchronize
 *              the startup of each AP with the host.
 *
 *****/
main( int argc, char *argv[], char *args[] )
{
    int pid;

    /* Get pid, split app shared memory address, and make noncache */
    pid = bolt_getpid ();

    printf ("Pl/AP0: pid = 0x%x \n", pid);
    fflush (stdout);

    hostShmPIAP0 = (struct rbfData *)get_share_mem_addr();

    printf ("Pl/AP0: Split app shared memory address = 0x%x \n", hostShmPIAP0);
    fflush (stdout);

    mprotect ((caddr_t)hostShmPIAP0, sizeof(struct rbfData), UCACHE);

    /* Set split app shared memory buffer pointers */
    hostShmPIAP0->Status = 0;

    /* Setup signal catcher */
    i860_signal (SIGUSR1, (void *) signalCatcher);

    /* release the host */
    host_kill (pid, SIGUSR1);

    /* wait for the host to signal */
    if (waitForSignal() == 0) {
        printf ("Pl/AP1: timeout occurred waiting for SIGUSR1 signal.\n");
        exit (1);
    }

    /* APO processing */

```

racoon:/mnt/avds/v5.0/sky/platform1/ap0/Plap0.c

Wed Apr 23 14:32:20 1997

```

/* Begin simple test */
/* Wait for reset release from host */
printf ("P1/AP0: Waiting for start signal from host\n");
fflush (stdout);
while (hostShmP1AP0->Status != SYSTEM_START) usleep(250);
printf ("P1/AP0: Started!! \n");
fflush (stdout);
rbf_recog( argc, argv);
printf ("P1/AP0: exiting \n");
fflush (stdout);

/* End simple test */
}

/*****
 *
 * Function: signalCatcher
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock
 * Arguments: None
 * Returns: N/A
 *
 * This function services the SIGUSR1 signal from the host.
 * Rev History: 4/19/95 - (mep) Initial design.
 *****/
void signalCatcher()
{
    waitFlag = 0;
}

/*****
 *
 * Function: waitForSignal
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock
 * Arguments: None
 * Returns: 1 - signal received, 0 - time out occurred
 *
 * This function waits for the SIGUSR1 signal from the host
 * Rev History: 4/19/95 - (mep) Initial design.
 *****/
int waitForSignal ()
{
    int timeout = 100; /* wait for approx. 100 secs. */
    waitFlag = 1;
    while (waitFlag && timeout-->0)
        sleep(1);
    if (timeout == 0)
        return (0);
    else
        return (1);
}

/*****
 *
 * Function: getFV
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock
 * Arguments: None
 * Returns: void
 *
 * This function stores the FV's for testing the host-P1/AP0 interface
 * Rev History: 8/8/95 - (mep) initial design.
 *****/
void getFV()
{
    float *FVindex;
    int loop=10;
    FVindex = FVarray;
    printf("P1/AP0: getFV: FVarray at 0x%x, VME 0x%x\n",
           FVarray, apv2vme(FVarray));
    while (loop || (hostShmP1AP0->Status != AVDS_EXIT)) {
        if (hostShmP1AP0->Status == DATA_SENT) {
            hostShmP1AP0->Status = PLATFORM_BUSY;
            printf("P1/AP0: getFV: FVindex=0x%x, frame=%d, loop=%d\n",
                   FVindex, hostShmP1AP0->FrameNum,
                   loop);
            memcpy(FVindex, hostShmP1AP0, sizeof(float)*64*8);
            FVindex += 64*8;
            loop--;
        }
        hostShmP1AP0->Status = RBF_DONE;
    }
}

```

raccoon:/mnt/avds/v5.0/sky/platform1/ap0/P1ap0.c

Wed Apr 23 14:32:26 1997

```

/* * SCCS Version control header for %M% rev. %R%
* * Created from: %P% %I%
* * Current Date: %D% %T% Last Change: %E%
*/
FILE NAME      : %M%.c
IDENTIFICATION : UNIX
LIBRARY        : n/a
PURPOSE:
Static pattern classification using weighed sums of Gaussian radial functions.
PROCESSING:
(Describe how this module accomplishes its purpose.)
NOTES:
GLOBALS AFFECTED: No external globals.
-----Revision History-----
Date      | Author      | Reason for revision
19 AUG 93 | William Y. Huang | baseline
16 JUL 94 | William Y. Huang | implement streams
4 AUG 94  | William Y. Huang | Weighted training vector
18 AUG 94 | William Y. Huang | implement Laplacian
24 JAN 95 | William Y. Huang | Half data (for 32->16 experiments)
19 MAY 95 | Rick Snover    | Cosmetic and standardization changes.
*/
/* * SCCS Version control header
* * %Z% %M% %R% %Z%
* * Created from: %P% %I%
* * Current Date: %D% %T% Last Change: %E%
*/
#define SCOSid "%Z% %M% %R% %D%"
#define NNETS 8

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <string.h>
#include "hostshm.h"

#ifdef DEBUG
#define DPRINT0(F)      fputs((F), stderr); fflush( stderr );
#define DPRINT1(F,X)    fprintf(stderr, (F), (X)); fflush( stderr );
#define DPRINT2(F,X,Y)  fprintf(stderr, (F), (X), (Y)); fflush( stderr );
#define DPRINT3(F,X,Y,Z) fprintf(stderr, (F), (X), (Y), (Z)); fflush( stderr );
#else
#define DPRINT0(F)
#define DPRINT1(F,X)
#define DPRINT2(F,X,Y)
#define DPRINT3(F,X,Y,Z)
#endif

```

```

typedef struct
{
    int      nstreams;
    int      *inputdim;
    int      *snnodes;
    int      nnodes;
    int      outdim;
    float    **centloc;
    float    **spread;
    float    **weight;
    unsigned long ntrain;
    float    **m;
    float    **partial_w;
    float    *rbfout;
    float    *output;
    RBF;
}

/* * spread factor in a radial basis function is analogous to the
* * variance in a Gaussian pdf function
static float spread_factor = 1.0;

/* * FUNCTION DECLARATIONS */
RBF
*allocate_rbf ();
float
*rbf_nodes_eval ();
static float
*matrix ();
void
Usage ();
void usleep (unsigned microseconds);

/* * FUNCTION DEFINITIONS */
FUNCTION NAME : allocate_rbf
FACILITY      : UNIX
IDENTIFICATION :
PURPOSE:
Allocate an rbf model, initializing spread, m and partial_w with (float *) NULL
INPUTS
OUTPUTS
PROCESSING:
Use malloc() to create space for an rbf, and initialize dimensionality values
NOTES:
GLOBALS AFFECTED:
Date      | Author      | Revision History
19 AUG 93 | William Y. Huang | Reason for revision
-----
*
RBF
allocate_rbf (inpdim, nnodes, nclass, nstreams)
int
*inpdim;
int
*nnodes;

```

racocon:/mnt/avds/v5.0/sky/platform1/ap0/rbf.c

Wed Apr 23 14:32:26 1997

```

int nclass;
int nstreams;
int i;
RBF *rbf;
/* allocate all memory necessary to store RBF parameters */
rbf = (RBF *) malloc ((unsigned) sizeof (RBF));
rbf->inputdim = (int *) malloc (sizeof (int) * nstreams);
rbf->snodes = (int *) malloc (sizeof (int) * nstreams);
rbf->outputdim = nclass;
rbf->nstreams = nstreams;
rbf->nnodes = 1;
/* start with one for the unity
* node */
for (i = 0; i < nstreams; i++)
{
    rbf->inputdim[i] = inputdim[i];
    rbf->snodes[i] = nnodes[i];
    rbf->nnodes = rbf->nnodes + nnodes[i];
}
/* BUILD MATRICES..... */
rbf->centloc = (float **) malloc (sizeof (float **) * nstreams);
rbf->spread = (float **) malloc (sizeof (float **) * nstreams);
for (i = 0; i < nstreams; i++)
{
    rbf->centloc[i] = matrix (nnodes[i], inputdim[i]);
    rbf->spread[i] = (float **) NULL;
}
rbf->weight = matrix (nclass, rbf->nnodes);
rbf->rbfout = (float *) malloc ((unsigned) sizeof (float) * rbf->nnodes);
rbf->output = (float *) malloc ((unsigned) sizeof (float) * nclass);
if (rbf->centloc == NULL || rbf->weight == NULL)
    return ((RBF *) NULL);
rbf->m = rbf->partial_w = (float **) NULL;
return (rbf);
}

/*
FUNCTION NAME : rbf_eval
FACILITY : UNIX
IDENTIFICATION : rbf
PURPOSE:
Compute output of an rbf model given an input vector
INPUTS : input vector and an rbf model
OUTPUTS : vector of outputs (same as rbf->output)
PROCESSING: vector of rbf outputs (hidden layer activation)
NOTES:
GLOBALS AFFECTED:
Date | Author | Revision History
19 AUG 93 | William Y. Huang | Reason for revision
12 SEP 93 | William Y. Huang | baseline
float *
rbf_eval (input, rbf, rbfout)
*/
static float **
matrix (m, n)
int m;
int n;
{
    int i;
    float **mat;
    /* indexing variable */
    /* temporary matrix */
    mat = (float **) malloc ((unsigned) (sizeof (float *) * m));
    if (mat == NULL)
        return (mat);
    return (mat);
}
/* array index 0 for numerical recipes */
mat = &mat[-1];
/*
* assign one continuous block of memory for the array, starting with
* the array's first element
mat[1] = (float *) malloc ((unsigned) (sizeof (float) * m * n));
if (mat[1] == NULL)
{
    free ((char *) mat);
    return ((float **) NULL);
}
/* force indexing to start at 1 */
mat[1] = &mat[1][-1];
/*
* assign memory location for all other rows to be in continuous memory
* locations
for (i = 2; i <= m; i++)
    mat[i] = &mat[1 - 1][n];
return (mat);
}

```

racoon:/mnt/avds/v5.0/sky/platform1/ap0/rbf.c

```

register float *input;
RBF
float
{
    register float *weights;
    register float *output;
    register float x;
    register int nnodes;
    register int i;
    register int j;

    input = rbf_nodes_eval (input, rbf);
    /* compute hidden layer, input now
       * assigned to output of hidden
       * layer */

    /* top layer transfer matrix */
    /* output vector */
    /* number of hidden nodes */
    /* hidden node vector */

    /* the top layer is essentially a matrix multiplication */
    for (i = rbf->outdim; i--;)
    {
        x = *weights++;
        /* for the first mult. of inner
           * product */
        for (j = nnodes - 1; j--;)
            x = x + *weights++ * *input; /* loop count minus 1 for identity */
        /* notice the identity (first element) is skipped */

        *output++ = x;
        input = input - nnodes + 1;
        /* store output */
        /* re-initialize the vector of
           * hidden */
        /* layer outputs for the computation */
    }

    return (&output[-rbf->outdim]);
}

/* FUNCTION NAME : rbf_nodes_eval
   FACILITY : UNIX
   IDENTIFICATION : rbf
   PURPOSE:
   Evaluate the rbf (hidden) node distances.
   INPUTS : input vector, (pre-initialized) rbf model
   OUTPUTS : vector output (same as rbf->rbfout)
   PROCESSING:
   Euclidean distance computations
   NOTES:
   GLOBALS AFFECTED:

   Date | Author | Revision History-----
   19 AUG 93 | William Y. Huang | Reason for revision
   -----
   float
   rbf_nodes_eval (input, rbf)
   register float *input;
   RBF
   *rbf;
   /* trained (with centroids loaded) */

```

racoon:/mnt/avds/v5.0/sky/platform/lap0/rbf.c

Wed Apr 23 14:32:26 1997

```

/* compute distances weighed by the spreads */
x = (float) 0.0;
for (j = inpdim[stream]; j > 0; --j)
{
    y = *input++ - *centptr++; /* difference */
    x = x + ((float) 0.5) * y * y / /* Gaussian */
    (*spdr++ * spread_factor * spread_factor);
}

x = x / inpdim[stream]; /* normalization */

/* Gaussian requires exponentiation */
if (x > 20)
    *outptr++ = (float) 0.0; /* catch overflow */
else
    *outptr++ = exp (-x);

/* re-initialize the input vector for the next iteration */
input = input - inpdim[stream];
}

/* shift input vector to the next stream */
input = input + inpdim[stream];
}

return (outptr - rbf->nnodes); /* return output pointer */
}

/*
FUNCTION NAME : rbf_read_bin_model()
FACILITY : UNIX
IDENTIFICATION : rbf
PURPOSE:
Read an rbf model written by rbf_write_bin_model()

INPUTS : file pointer
OUTPUTS : binary rbf model
PROCESSING:
Use C Library functions fwrite() and fread()

NOTES:
This routine is probably unnecessary since ascii files are stored fairly
efficiently after compression

GLOBALS AFFECTED:

rbf_read_bin_model()
Date | Author | Revision History-----
19 AUG 93 | William Y. Huang | baseline
-----
RBF *
rbf_read_bin_model(file)
FILE *file;
{
    int *inpdim; /* inpdim[j] = dimension of j-th
                  * stream */
    int *snodes;
    outdim = -1;
    nstreams;
    i;
    j;
    *rbf = (RBF *) NULL;
    /* rbf model, NULL flags an
    un-initialized condition */

    if (1 != fread (&nstreams, sizeof (int), 1, file))
    {
        fprintf (stderr, "binary weights file read error\n");
        exit (-1);
    }
    inpdim = (int *) malloc (sizeof (int) * nstreams * 2);
    snodes = &inpdim[nstreams];
    if (nstreams != fread (inpdim, sizeof (int), nstreams, file) ||
        nstreams != fread (snodes, sizeof (int), nstreams, file) ||
        1 != fread (&outdim, sizeof (int), 1, file))
    {
        fprintf (stderr, "binary weights file read error\n");
        exit (-1);
    }
    rbf = (RBF *) allocate_rbf (inpdim, snodes, outdim, nstreams);
    if (rbf == NULL)
        return (rbf);

    for (i = 0; i < nstreams; i++)
        fread (&rbf->centloc[i][1], sizeof (float),
            rbf->snodes[i] * inpdim[i], file);

    fread (&rbf->weight[1][1], sizeof (float), rbf->nnodes * outdim, file);
    for (i = 0; i < nstreams; i++)
    {
        fread (&j, sizeof (int), 1, file);
        if (j == 1)
        {
            rbf->spread[i] = matrix (snodes[i], inpdim[i]);
            fread (&rbf->spread[i][1][1], sizeof (float),
                rbf->snodes[i] * inpdim[i], file);
        }
        free (inpdim);
        return (rbf);
    }

    static char *usage = "
Usage: %s <options>\n\
-recog savefile load rbf from savefile and do recognition run\n\
-factor spread_factor (def=%g)\n\
";

    /*
FUNCTION NAME : Usage()
FACILITY : UNIX
IDENTIFICATION : rbf
PURPOSE:
Display usage help and terminate the program.

```

racoon:/mnt/avds/v5.0/sky/platform1/ap0/rbf.c

```

INPUTS      : a string to be displayed
OUTPUTS     : writes the input string, usage[] text, max_ratio, and
              spread_factor to the console
PROCESSING  :
NOTES:
GLOBALS AFFECTED:
-----Revision History-----
Date      | Author      | Reason for revision
19 MAY 95 | Rick Shover | baseline
-----
void
Usage (str)
char      *str;
{
    fprintf (stderr, usage, str, spread_factor);
    exit (0);
}

/* this should already be defined ..... but just in case ... */
#define MAX_INPUT
#define MAX_INPUT 256
#endif

/*
FUNCTION NAME : main()
FACILITY     : UNIX
IDENTIFICATION : rbf
PURPOSE:
Control the execution of the RBF.
INPUTS      : argc, argv
OUTPUTS     :
NOTES:
GLOBALS AFFECTED:
-----Revision History-----
Date      | Author      | Reason for revision
24 JAN 93 | William Y. Huang | baseline
-----
int
rbf_recog (argc, argv)
int      argc;
char     **argv;
{
    /* radial basis functions */
    /* number of outputs */
    noutputs[NNETS];
    /* number of inputs */
    ninputs[NNETS];
    /* file pointer used for both
    model and data input */
    *inpfile;
    /* input vector */
    *invec[NNETS];
    /* vector of outputs */
    *outvec;
    /* vector of rbf
    (hidden) layer output */
    i;
    /* indexing variable */
    j;
    /* indexing variable */
    k;
    /* network indexing variable */
    neti;
    /* temporary floating storage */
    max;

```

```

int      skipchan=0;
int      skip_start=0;
int      skip_end=0;
int      index;
int      max_vect_size=0;
char      *chanptr;
float     *vector=(float *)NULL;
float     *fvectptr;
char      number[10];

for (neti=0; neti<NNETS; neti++)
{
    rbf[neti] = (RBF *)NULL;
    noutputs[neti] = 0;
    ninputs[neti] = 0;
}

/* PROCESS COMMAND LINE arguments */
for (i = 2; i < argc && argv[i][0] == '-'; i++) {
    if (!strcmp (argv[i], "-factor")) {
        if (i != sscanf (argv[i+1], "%f", &spread_factor)) {
            fprintf (stderr, "error with the -factor option\n");
            usage (argv[0]); /* Never returns */
        }
    }
    else if (!strcmp (argv[i], "-recog")) {
        if ((chanptr = strstr (argv[i+1], "rbf" )) == NULL)
            fprintf (stderr,
                    "error: weight file name prefix must be rbf\n");
        exit(1);
    }
}

for (neti=0; neti<NNETS; neti++)
{
    chanptr[4] = (char)(neti+1 + '0');
    if (!inpfile = fopen (argv[i], "r")) == NULL)
        fprintf (stderr, "error reading model input file %s\n", argv[i]);
    exit (-1);
}

rbf[neti] = rbf_read_bin_model (inpfile); /* read_model takes care
of streams */
fclose (inpfile);
if (rbf[neti] == NULL)
    exit (-1);
noutputs[neti] = rbf[neti]->outdim; /* set up local variables */
printf ("P1/AP0: Net %d: %d Hidden Nodes, %d Outputs\n",
        neti+1, rbf[neti]->nnodes - 1, noutputs[neti]);
}

/* Sum input dimensions of channels 2 through 8 from net 1 */
max_vect_size = 0;
for (neti = 0; neti < rbf[0]->nstreams; neti++)
    max_vect_size += rbf[0]->inputdim[neti];

/* Add input dimension of channel 1 from net 2 to get maximum
vector size */
max_vect_size += rbf[1]->inputdim[0];

```

racocon:/mnt/avds/v5.0/sky/platform1/ap0/rbf.c

Wed Apr 23 14:32:26 1997

```

/* Create input vector array */
if ((vector = (float *) malloc( (unsigned) (max_vect_size *
    sizeof( float ) ) ) == (float *) NULL)
    {
        fprintf( stderr, "error: could not create input vector array\n" );
        exit( 0 );
    }
}
else {
    fprintf( stderr, "Unrecognized option: %s\n", argv[i] );
    Usage( argv[0] );
    /* Never returns */
}

for (neti=0; neti<NNETS; neti++)
{
    /* ninputs is the sum of the inpdims of all the streams */
    for (ninputs[neti] = i = 0; i < rbf[neti]->nstreams; i++)
        ninputs[neti] = ninputs[neti] + rbf[neti]->inputdim[i];
    invect[neti] = (float *) malloc( (unsigned) (sizeof( float )
        * ninputs[neti]) );
    if (invect[neti] == (float *) NULL)
    {
        fprintf( stderr, "error: could not create vector working space\n" );
        exit( 0 );
    }
}

/* tell the host that the rbf is ready */
hostShmPIAP0->Status = PLATFORM_READY;

while (hostShmPIAP0->Status != AVDS_EXIT)
{
    if (hostShmPIAP0->Status == DATA_SENT)
    {
        hostShmPIAP0->Status = PLATFORM_BUSY;

        /* normal data format processing */
        FVectPtr = &(hostShmPIAP0->FVect[0][0]);
        for (i = 0; i < max_vect_size; i++)
        {
            vector[i] = *FVectPtr++;
            printf( number, "%7.5f\n", vector[i] );
            vector[i] = atof( number );
        }

        for (neti=0; neti<NNETS; neti++)
        {
            if (neti == 0)
            {
                skip_start = neti * rbf[7]->inputdim[0];
                skip_end = skip_start + rbf[7]->inputdim[0] - 1;
            }
            else
            {
                index = rbf[neti-1]->inputdim[neti-1];
                skip_start = neti * index,

```

```

                skip_end = skip_start + index - 1;
            }
            for (i=0, j=0; i<max_vect_size; i++)
            {
                if ( i<skip_start || i>skip_end )
                    invect[neti][j++] = vector[i];
            }

            /* run through the hidden layer */
            outvec = rbf_eval( invect[neti], rbf[neti], &rbfout );

            /* sigmoidal output layer */
            /* this brings output to [0,1] for easier handling */
            for (j = 0; j < noutputs[neti]; j++)
                outvec[j] = 1.0 /
                    (1.0 + exp( -(outvec[j] - .5) * 4 ));

            /* GET THE RECOGNIZED CLASS as a MAX of the outvec's */
            k = 0;
            max = outvec[k];
            for (j = 0; j < noutputs[neti]; j++)
            {
                if (outvec[j] > max)
                {
                    k = j;
                    max = outvec[j];
                }
            }

            /* Output fault class */
            if (hostShmPIAP0->PrintFlag == DISPLAY)
            {
                printf( "PIAP0: RBF Net %d Class %d\n", neti+1, k );
                fflush( stdout );
            }

            /* Output activation energies */
            for (j = 0; j < noutputs[neti]; j++)
                hostShmPIAP0->FaultInd[neti][j] = outvec[j];

            /* for (neti=0; neti<NNETS; neti++) */
            hostShmPIAP0->Status = RBF_DONE;
        }
        else
            usleep( 250000 );
    }

    /* no error return value */
    return (0);
}

```

raccoon:/mnt/avds/v5.0/sky/platform/rap0/rbf.c

```

/*
 * SCCS Version control header for P1ap1.c rev. 1
 * Created from: /home/garrido/v10.0/platform1/ap1/sccs/s.P1ap1.c 1.2
 * Current Date: 3/26/96 14:27:11 Last Change: 3/26/96
 */
*****
 *
 * File: P1ap1.c
 * Program: P1ap1prog
 * OS: SKYvec v3.5
 * Execute: N/A
 * Link list:
 *
 * This program provides the framework for split application with shared
 * memory.
 *
 * Rev History: 09/08/95 - mep
 */
*****
#define MAIN_PROG
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/boltprotos.h>
#include <sky/bolt_ipc.h>
#include <sky/bolt_shm.h>
#include <memory.h>
#include "hostshm.h"

#define CACHE 0x47
#define UCACHE 0x5f /* Sets write through and cache disable */
*****
 * external declarations
 *
*****
void aa(int argc, char **argv);
int bolt_getpid();
char *get_share_mem_addr();
void usleep(unsigned seconds);
void sleep(unsigned seconds);
int apv2vme(void *);
*****
 * forward declarations
 *
*****
void signalCatcher();
void getFV();
*****
 * module level variables
 *

```

```

\*****
int waitFlag;
float FVarray[64*8*10];
/*****
 *
 * Function: main
 * OS: SKYvec v3.5
 * Target: SKYbolt Shamrock API
 * Arguments: None
 * Returns: N/A
 *
 * This function is the "main" routine runs the Platform 1, API
 * application.
 *
 * Rev History: 3/15/95 - (dki) Initial design.
 *              4/19/95 - (mep) restructured shared memory framework
 *                      and setup signal handlers to synchronize
 *                      the startup of each AP with the host.
 *              7/18/95 - (rcg, dki) - main application code
 *                      moved to function aa().
 *              3/26/96 - (mep) potential bug fix - corrected sizeof in
 *                      mprotect instruction.
 */
*****
main( int argc, char *argv[], char *argc[] )
{
    int pid;

    /* Get pid, split app shared memory address, and make noncache */
    pid = bolt_getpid();

    printf("P1/API: pid = 0x%x\n", pid);
    fflush(stdout);

    hostShmP1API = (struct aaData *)get_share_mem_addr();
    printf("P1/API: Split app shared memory address = 0x%x\n", hostShmP1API);
    fflush(stdout);
    mprotect((caddr_t)hostShmP1API, sizeof(struct aaData), UCACHE);

    /* Set split app shared memory buffer pointers */
    hostShmP1API->Status = 0;

    /* Setup signal catcher */
    i860_signal(SIGUSR1, (void *) signalCatcher);

    /* release the host */
    host_kill(pid, SIGUSR1);

    /* wait for the host to signal */
    if (waitForSignal() == 0) {

```

raccoon:/mnt/avds/v5.0/sky/platform1/ap1/P1ap1.c

Wed Apr 23 14:32:51 1997

```

printf ("P1/API: timeOut occurred waiting for SIGUSR1 signal.\n");
fflush (stdout);
exit (1);
}

/* API processing */
/* Begin simple test */

/* Wait for reset release from host */
printf ("P1/API: Waiting for start signal from host\n");
fflush (stdout);
while (hostShmP1API->Status != SYSTEM_START) usleep(250);
printf ("P1/API: Started!! \n");
fflush (stdout);
aa( argc, argv);

printf ("P1/API: exiting \n");
fflush (stdout);

/* End simple test */
}

/*****
*
* Function: signalCatcher
* OS: SKYvec v3.5
* Target: SKYbolt Shamrock
* Arguments: None
* Returns: N/A
*
* This function services the SIGUSR1 signal from the host.
* Rev History: 4/19/95 - (mep) initial design.
*****/

void signalCatcher()
{
    waitFlag = 0;
}

/*****
*
* Function: waitForSignal
* OS: SKYvec v3.5
* Target: SKYbolt Shamrock
* Arguments: None
* Returns: 1 - signal received, 0 - time out occurred
*
* This function waits for the SIGUSR1 signal from the host
*****/

int waitForSignal ()
{
    int timeOut = 100; /* wait for approx. 100 secs. */
    waitFlag = 1;
    while (waitFlag && timeOut--)
        sleep(1);
    if (timeOut == 0)
        return (0);
    else
        return (1);
}

/*****
*
* Function: getFV
* OS: SKYvec v3.5
* Target: SKYbolt Shamrock
* Arguments: None
* Returns: void
*
* This function stores the FV's for testing the host-P1/API interface
* Rev History: 8/8/95 - (mep) initial design.
*****/

void getFV()
{
    float *FVindex;
    int loop=10;
    FVindex = FVarray;
    printf("P1/API: getFV: FVarray at 0x%x, VME 0x%x\n\n",
        FVarray, apv2vme(FVarray));
    while (loop || (hostShmP1API->Status != AVDS_EXIT)) {
        if (hostShmP1API->Status == DATA_SENT) {
            hostShmP1API->Status = PLATFORM_BUSY;
            printf("P1/API: getFV: FVindex=0x%x, frame=%d, loop=%d\n",
                FVindex, hostShmP1API->FrameNum,
                loop);
            memcpy(FVindex, hostShmP1API, sizeof(float)*64*8);
            FVindex += 64*8;
            loop--;
            hostShmP1API->Status = RBF_DONE;
        }
    }
}

```

racocon:/mnt/avds/v5.0/sky/platform1/ap1/P1ap1.c

```

/* SCS Version control header for autoassoc.c rev. 1
 * Created from: /home/garriido/v10.2/platform1/ap1/SCCS/s.autoassoc.c 1.2
 * Current Date: 4/23/96 09:11:47 Last Change: 4/23/96
 */
/* autoassoc.c
 * last modified - 24 Jan 96 - dki
 * single precision version
 * weight file is single precision binary
 * handles 8 autoassociator nets
 * 7 of 8 input channels
 * based on label.c, net.c, and pat.c received from Siemens
 * Program to compute and print the MSE for each feature vector
 * Arguments are network-description-file and pattern-file
 */
/* Copyright 1995 by Siemens Corporate Research, Inc.
 * Written by Thomas Petsche
 * Last change July 1995
 */

```

```

#include "autoassoc.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include "hostshm.h"

```

```
void usleep (unsigned microseconds);
```

```

#define TIME
float dttime();
#endif

```

```

#define ERROR(S) fprintf( stderr, "%s: %s", argv[0], S );
#define NNETS 8
#define MAXINP 8*64
#define CHANSIZE 64

```

```

/*-----
FUNCTION NAME : aa()
FACILITY : UNIX
IDENTIFICATION : Autoassociator
PURPOSE : Controls the execution of the eight auto-correlation
          nets on a Sky Shamrock platform.

INPUTS : argc, argv - command line options.
        hostShmAPI1 - a 512 element feature vector.
OUTPUTS : hostShmAPI1 - elements of the host shared memory output
        data structure are altered.

PROCESSING : PARSE the command line argument.
            CREATE AA nets from network description files.
*/

```

```

SIGNAL host that the AA platform is ready.
DETERMINE channel features skip region for each net.
LOOP until exit signal is received from host.
IF the host signals feature vectors ready,
COPY the 512 element feature vector into the input
vector array.
LOOP through the eight AA nets.
COPY feature vector into input array excluding
feature vector skip regions.
EVALUATE the MSE (MSE())
IF the DISPLAY flag is set,
PRINT the MSE for the net.
STORE the MSE in the host shared memory output
data structure.
SIGNAL host that the AA output Mean Squared Error
values are available for the eight AA nets.
ELSE
SLEEP for 250 milliseconds.
RETURN

```

NOTES

: There are eight AA nets. Each net was trained with data that excluded one of the eight accelerometer channels. After the complete 512 element feature vector is read in, each net is given a 448 element subset of the vector to work on. This feature vector subset is formed by omitting the sixty-four elements corresponding to the accelerometer channel that the selected net is required to ignore. The eight net AA structure allows the detection system to function properly in the event that a non-critical accelerometer were to fail.

A Mean Squared Error (MSE) value is output by each net. MSEs are floating point values ranging from 0 to 1.0.

Date	Author	Revision History
24 JAN 96	DKI	Reason for revision
		Single precision (float). Capable of handling 100 hidden layers.
04 OCT 95	DKI	Sky Shamrock split application conversion. Unnecessary code stripped out.
01 JUL 95	Thomas Petsche	baseline

```

void aa( int argc, char **argv )
(
    NET *n(NNETS);
    float FVnet[1000];
    int i, j, k, m;
    int skip_count;
    int skip_start[NNETS], skip_end[NNETS];
    char *netfile;
    char *chanptr;
    float *vector;
    float dt, time[2];
    #ifdef TIME
    float
    #endif
)

```

racoon:/mnt/avds/v5.0/sky/platform1/ap1/autoassoc.c

Wed Apr 23 14:33:08 1997

```

netfile = argv[6];
if ((chanptr = strstr( netfile, ".100.fin" )) == NULL)
{
    fprintf( stderr,
        "error: weight file name extension must be .100.fin\n" );
    exit( 1 );
}

for(i=0; i<NNETS; i++)
{
    /* Create AA nets based on definition files */
    chanptr[i] = (char)i + '0';
    printf("Network definition file: %s\n", netfile);
    if( (n[i] = ReadNet(netfile)) == NULL )
    {
        ERROR( "Could not read network description file\n" );
        exit(-1);
    }

    if(n[i]->NeuronsInLayer[n[i]->InputLayer] !=
        n[i]->NeuronsInLayer[n[i]->OutputLayer])
    {
        ERROR( "Size of input and output layers do not match.\n" );
        fprintf( stderr, "Not a valid autoassociator.\n" );
        exit(-1);
    }
}

/* Determine skip count */
skip_count = CHAN_SIZE;

/* for each net, determine start and end of channel to be skipped */
for (i=0; i<NNETS; i++)
{
    skip_start[i] = i * skip_count;
    skip_end[i] = skip_start[i] + (skip_count-1);

    /* signal the host that the network config files have been read */
    hostShmPIAP1->Status = PLATFORM_READY;
    while (hostShmPIAP1->Status != AVDS_EXIT)
    {
        if (hostShmPIAP1->Status == DATA_SENT)
        {
            #ifdef TIME
                dt = (float)dtm(time);
            #endif
            hostShmPIAP1->Status = PLATFORM_BUSY;

            /* point to feature vector */
            vector = &(hostShmPIAP1->FVect[0][0]);
            /* for each feature vector, process all nets */
            for (j=0; j<NNETS; j++)
            {
                m = 0;
            }
        }
    }
}

/* determine net input vector */
for (k = 0; k < MAXINP; k++)
    if ( (k < skip_start[j]) || (k > skip_end[j]) )
        FVnet[m++] = (float)vector[k];
    hostShmPIAP1->MSEout[j] = MSE(n[j], FVnet, 1);

if (hostShmPIAP1->PrintFlag == DISPLAY)
{
    printf( "PIAP1: AA Net %d MSE %7.5f\n", j,
        hostShmPIAP1->MSEout[j] );
    fflush( stdout );
}

hostShmPIAP1->Status = AA_DONE;

#ifdef TIME
    dt = (float)dtm(time);
    fprintf( stderr, "AA time %3f \n", dt );
    fflush( stderr );
#endif
    else
        usleep( 250000 );
    return;
}

SyntaxError(prg)
char * prg;
{
    fprintf( stderr, "\nSyntax: %s net-desc-file pattern-file\n", prg );
    exit(-1);
}

/******
/* Read a feed forward network description from a file
/* Format is the same as that written by WriteNet.
/* Number of layers, (integer)
/* Number of neurons in each layer (integers)
/* Threshold and input weights for each neuron (floats)
/* Arguments:
/* fname - name of the file to read from
/* MUST BE IN CORRECT FORMAT!!!
/* Return values:
/* pointer to NET if successful
/* NULL if unsuccessful (nothing read or error in input)
/******
#define INPUT_ERROR fprintf( stderr, "Error in reading network description file:
    .
)
NET *

```

racocon:/mnt/avds/v5.0/sky/platform1/ap1/autoassoc.c

```

ReadNet( char *fname)
{
    FILE *fp;
    int i,k;
    int layers, *NIL;
    NET *n;

    if( (fp=fopen( fname, "r" )) == NULL )
    {
        fprintf( stderr, "Can't open %s for reading.\n", fname );
        return( NULL );
    }

    /* Read the network description parameters */
    /* The number of layers */
    if( (!fread( &layers, sizeof( int ), 1, fp ))
        INPUT_ERROR;
        fprintf( stderr, "\tBad 'Layers' entry.\n" );
        return( NULL );
    )
    NIL = (int *)calloc( layers, sizeof(int) );

    /* The number of neurons in each layer */
    if( (!fread( &NIL[0], sizeof( int ), layers, fp ))
        INPUT_ERROR;
        fprintf( stderr, "\tBad entry\n" );
        return( NULL );
    )

    /* allocate space for the network */
    if( n = InitNet( layers, NIL ) == NULL )
    {
        free( NIL );
        fprintf( stderr, "\tNetwork initialization failed\n" );
        return( NULL );
    }

    /* Threshold and input weights for each neuron (floats) */
    for( k=1; k<n->layers; k++ )
    {
        if( (!fread( &(n->thresh[k][1]), sizeof( float ), 1, fp ))
            INPUT_ERROR;
            fprintf( stderr,
                "\tBad 'threshold' entry for neuron %d in layer %d\n",
                k, i );
            return( NULL );
        )
        if( (!fread( &(n->w[k][1][0]), sizeof( float ),
            n->NeuronsInLayer[k-1], fp ))
            INPUT_ERROR;
            fprintf( stderr,

```

racoon:/mnt/avds/V50/sky/platform1/ap1/autoassoc

Wed Apr 23 14:33:08 1997

```

return(NULL);

if( (n->thresh = (float **)calloc(layers, sizeof(float *)) ) == NULL )
    return(NULL);
for( l=0; l<layers; l++)
    if( (n->thresh[l] = (float *)calloc(NIL[l], sizeof(float)) ) == NULL )
        return(NULL);
    return(n);
}

/* compute the mean square error between the desired
 * and actual outputs
 * Arguments:
 * n - pointer to the network to be used
 * pattern - pointer to an array of pattern (feature vectors) to be used
 * npats - the number of patterns to be evaluated
 * Returns:
 * the mean square error for all the patterns
 */
float
MSE( NET *n, float *pattern, int npats )
{
    double err, mse;
    int i;
    unsigned long OutputCnt=0;

    mse = 0;
    ForwardPass(n, pattern);
    for( i=0; i<n->NeuronsInLayer[n->OutputLayer]; i++ )
    {
        OutputCnt++;
        err = (double)(n->x[n->OutputLayer][i] - pattern[i]);
        mse += (err*err);
    }
    return( (float)sqrt(mse/OutputCnt) );
}

/* compute a single forward pass through the network
 * for the given pattern
 * Arguments:
 * n - a pointer to the network to be used
 * pattern - a pointer to the pattern (feature vector) to be used
 * Returns:
 * 1 on success
 */
int
ForwardPass( NET *n, float *pattern )
{
    int i, j, l;
    double sum;
    /* generic counters */

    for( i=0; i<n->NeuronsInLayer[n->InputLayer]; i++ )
    {
        n->x[n->InputLayer][i] = pattern[i];
    }

    /* execute the forward pass */
    for( l=n->InputLayer+1; l<n->OutputLayer; l++ )
    {
        for( i=0; i<n->NeuronsInLayer[l]; i++ )
        {
            sum = (double)(n->thresh[l][i]);
            for( j=0; j<n->NeuronsInLayer[l-1]; j++ )
                sum += (double)(n->w[l][i][j]*n->x[l-1][j]);
            n->x[l][i] = (float)F(sum, l, i);
        }
        return(l);
    }
}

```

racocon:/mnt/avds/v5.0/sky/platform1/ap1/autoassoc.c

```

/*****
 *
 * Program: AVDScomm
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRad, Code 535
 * email: philipm@nosc.mil
 *
 * This program provides serial communications between the NRP system
 * and the Teledyne "Regime Processor". The data output packet
 * (structure) is transmitted every second. The data input packet
 * (structure) is expected at a rate of 4 Hz.
 *
 * Rev History: 4/11/95 (MEP) - Initial design. Uses a timer to
 * signal output and determine input data packet rate.
 * Data starvation is declared at 16 seconds or
 * 64-250msec time intervals.
 * 9/05/95 (MEP) - design was changed to allow for data
 * to be received in bursts instead of at precise 250 msec
 * intervals.
 *****/

#include "vxWorks.h"
#include "ioLib.h"
#include "semLib.h"
#include "signal.h"
#include "sigLib.h"
#include "tyLib.h"
#include "timers.h"
#include ".../sky/include/message.h" /* structure definitions for messages */
#include ".../tda/vxPlayback/tda.h" /* added for answer processing */

#define PRINT_IN_MESSAGE
#define SAVE_OUTPUT_MESSAGE
#define SAVE_INPUT_MESSAGE

/* excluded facilities */
#if FALSE

#define PRINT_ANSWER
#define DEBUG
#endif

/*****
 *
 * macro constants
 *
 *****/

#define TIMEOUT_TICKS 64
#define TIMEOUT_250_mSEC (TIMEOUT_TICKS/4)
#define TIMEOUT_IN_SECS 0
#define TIMEOUT_IN_NSECS 250000000
#define TICKS_PER_SECOND 4

#define ANSWER_QUEUE_SIZE 5
#define ANSWER_INTERVAL (3*TICKS_PER_SECOND)
#define MIN_FR_TICKS (30*TICKS_PER_SECOND)

/*****
 *
 * forward declarations
 *
 *****/

void AVDSoutput();
void AVDSinput();
STATUS AVDScomm();
STATUS AVDSstop();

/*****
 *
 * status bit fields */
#define NO_ERROR 0
#define FIRST_MESSAGE 2
#define RP_NOT_SENDING_1 4
#define RP_NOT_SENDING_16 8
#define RP_RECV_ERROR 16
#define RP_SEND_ERROR 32
#define REGIME_INVALID 64

/*****
 *
 * answer queue structure
 *****/

struct ANSWER_REC {
    unsigned short answer; /* answer */
    unsigned char hour; /* answer time */
    unsigned char minute;
    unsigned char second;
};

/*****
 *
 * global variables
 *****/

int fd_out; /* file descriptor for output message */
int fd_in; /* file descriptor for input message */

int ttyb; /* serial device descriptor */
int ttyc; /* id for spawned input task */
int inputTask; /* semaphore for synchronization */
SEM_ID timeSem; /* id for timer */
timer_t tim; /* mem for input */
struct MESSAGE_IN_REC inMessage; /* 250 msec counter */
struct MESSAGE_OUT_REC outMessage; /* flag for termination */
int gate; /* flag for gate */
int validFRTicks = 0; /* flight regime valid counter */
int answer; /* FIFO index counter */
struct ANSWER_REC answerQ[ANSWER_QUEUE_SIZE]; /* FIFO queue array */

/*****
 *
 * forward declarations
 *
 *****/

void AVDSoutput();
void AVDSinput();
STATUS AVDScomm();
STATUS AVDSstop();

```

racocon:/mnt/avds/v5.0/AVDScomm/v2.0/AVDScomm.c

Wed Apr 23 14:33:51 1997

```

unsigned char calc_checksum (unsigned char *buffer, int count);
int checksum (unsigned char *buffer, int count);
void processAnswer();
void ttyprint(char *string);

/*****
 *
 * Function: AVDSoutput
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: None
 * Returns: None
 *
 * This function provides basic timing through the use of timers and
 * SIGNAL handling. Output packets are written to the serial device
 * 'ttyb' every four ticks (SIGALRM) through the use of the counter
 * 'timeTick'. Software semaphores are used to "pace" the input at
 * 250 msec intervals and generate timeout signals when required.
 *
 * Rev History: 4/11/95 - initial design. Executes every SIGALRM
 *                  12/6/95 - added regime and answer processing
 *
 *****/

void AVDSoutput ()
{
    struct MESSAGE_OUT_REC *outRec=&outMessage; /* pointer to output
    struct MESSAGE_IN_REC *inRec=&inMessage;      /* pointer to input
    int ttyStat;
    static unsigned char prFr;
    #ifdef SAVE_OUTPUT_MESSAGE
    int nb;
    #endif

    /* make semaphore available momentarily, then take back to provide
    /* synchronization for input task. Since the semaphore was
    /* initialized as empty, this give controls the timing
    semGive(timeSem);
    semTake(timeSem, WAIT_FOREVER);

    /* maintain flight regime counter (1/4 sec per tick)
    if ((inMessage.flightRegime < 10) && (inMessage.flightRegime == prFr))
    else
    validFrTicks = 0;
    prFr = inMessage.flightRegime;

    /* process the answers every ANSWER_INTERVAL
    if (! (timeTick & ANSWER_INTERVAL))
    processAnswer();

    /* increment 250 msec counter and output at interval specified
    if (! (timeTick++ & TICKS_PER_SECOND)) {
    if (timeTick < 4)
    outRec->status |= FIRST_MESSAGE;
    outRec->checksum = calc_checksum ((unsigned char *)outRec,

```

```

        sizeof(outMessage) - 1);
    nb = write (fd_out, (char *)outRec, sizeof(outMessage));
    if (nb != sizeof(outMessage)) {
        printf("Error writing to output message file. \n");
    }
    #endif

    /* save current messages (in/out) to shared memory */
    memcpy ((char *)MEM_COMM_POINTER_ADDR, (char *)inRec, sizeof(inRec)+sizeof(o
utRec));

    ttyStat = write (ttyb, (char *)outRec, sizeof(outMessage));
    outRec->status &= ~FIRST_MESSAGE;
    if (ttyStat != sizeof(outMessage)) {
        printf("AVDSoutput: ERROR transmitting message\n");
        outRec->status |= RP_SEND_ERROR;
    }
    else
        outRec->status &= ~RP_SEND_ERROR;

    #ifdef DEBUG
    printf ("AVDSoutput: writing data\n");
    printf ("%s\n", outRec->status);
    #endif
}

/*****
 *
 * Function: AVDSinput
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: None
 * Returns: None
 *
 * This function waits for semaphore release and then attempt to read
 * from the serial device 'ttyb'. If no input data is available after
 * TIMEOUT_TICKS data starvation is declared. Partial messages are
 * discarded and the buffer is flushed to recover from a reset that
 * occurred in mid-message.
 *
 * Rev History: 4/11/95 - initial design. Executes every SIGALRM
 *                  signal(250 msec) through semaphore release.
 *****/

void AVDSinput ()
{
    struct MESSAGE_IN_REC *inRec=&inMessage; /* pointer to input
    int nb;
    int timeOut;
    int ttyStat;
    int deadCount=0;
    int reprintHeader = TRUE;
    #ifdef PRINT_IN_MESSAGE
    char printstring[80];
    char eol[2] = { '\n', '\0' };

```

racocon:/mnt/avds/v5.0/AVDScomm/v2.0/AVDScomm.c

Wed Apr 23 14:33:51 1997

```

#endef
while (1) {
    timeout = TIMEOUT_TICKS;
    nb = 0;
    while (timeout-->0) {
        /* check if any data in input buffer
        if ((ttyStat = ioctl(ttyb, FIONREAD, (int)&nb)) == ERROR)
            printf("ERROR accessing receive buffer length /cyco/2\n");
        #ifdef DEBUG
            printf ("AVDSInput: nb=%d, timeout = %d\n", nb, timeout);
            printf ("%d, %d, %d\n", nb, nb*sizeof(InMessage), timeout);
        #endif
        /* entire message(s) received
        if ((nb*sizeof(InMessage)) == 0) && (nb != 0))
            break;
        /* partial message received
        if (nb != 0)
            printf("ERROR flushing receive buffer on /cyco/2");
        /* wait for synchronization (AVDSoutput to release the semaphore)
        semTake(&timeSem, WAIT_FOREVER);
        semGive(&timeSem);
        /* reset the valid regime counter if RESET_REGIME_COUNTER has elapsed */
        if (timeout < RESET_REGIME_COUNTER)
            validFRTicks = 0;
    }

    /* log error if no data available, else read buffer and process
    if (timeout < 0) {
        outMessage.status |= RP_NOT_SENDING_16;
        deadCount++;
        printf("AVDSInput: ERROR - NO INPUT FOR %d SECS, COUNT=%d\n",
            TIMEOUT_TICKS/TICKS_PER_SECOND,
            deadCount);
        reprintHeader = TRUE;
    }
    else {
        outMessage.status &= -RP_NOT_SENDING_16;
        deadCount = 0;
        nb = read(ttyb, (char *)inRec, sizeof(InMessage));
        if (checksum((unsigned char *)inRec, sizeof(InMessage)) != 0) {
            printf ("nAVDSInput: Checksum ERROR.\n");
            outMessage.status |= RP_RECV_ERROR;
        }
        else {
            outMessage.status &= -RP_RECV_ERROR;
            outMessage.year = InMessage.year;
            outMessage.month = InMessage.month;
            outMessage.day = InMessage.day;
            outMessage.hour = InMessage.hour;
            outMessage.minute = InMessage.minute;
            outMessage.second = InMessage.second;
            #ifdef SAVE_INPUT_MESSAGE
                nb = write (fd_in, (char *)inRec, sizeof(InMessage));
                if (nb != sizeof(InMessage)) {
                    printf("Error writing to input message file. \n");
                }
            #endif
        }
    }
}
Function: AVDScomm

```

Function: AVDSComm
OS: VxWorks v5.1.1

racoon:/mnt/avds/v5.0/AVDScomm/v2.0/AVDScomm.c

Wed Apr 23 14:33:51 1997

```

* Target: Motorola MVME-166
* Author: M. E. Phillips, NRad, Code 535
* Arguments: None
* Returns: status
*
* This function is the "main" routine that kicks (spawns) the input
* process, programs the real-time clock, creates the signal handler.
* Startup synchronization is performed here. Once all processes are
* initialized, the routine waits until a flag is posted indicating
* to stop. "sigsuspend" is used to minimize overhead between SIGALRM
* signals.
*
* Rev History: 4/11/95 - initial design.
*
*****
STATUS AVDScomm()
{
    int ttyStat;
    struct itimerspec timeCtl;
    struct sigaction signalAction;

    /* open tty device, set appropriate controls and flush both input and
    /* output buffers.
    printf("AVDScomm: opening device /tyCo/1\n");
    if ((ttyb = open ("/tyCo/1", O_RDWR, 0666)) == ERROR) {
        printf("ERROR opening device /tyCo/1\n");
        return (ERROR);
    }

    printf("AVDScomm: setting baud rate on /tyCo/1\n");
    if ((ttyStat = ioctl(ttyb, FIOBAUDRATE, 19200)) == ERROR) {
        printf("ERROR setting baud rate on /tyCo/1\n");
        return (ERROR);
    }

    printf("AVDScomm: setting raw data mode on /tyCo/1\n");
    if ((ttyStat = ioctl(ttyb, FIOSETOPTIONS, OPT_RAW)) == ERROR) {
        printf("ERROR setting raw data mode on /tyCo/1\n");
        return (ERROR);
    }

    printf("AVDScomm: flushing input buffer on /tyCo/1\n");
    if ((ttyStat = ioctl(ttyb, FIOFLUSH, 0)) == ERROR) {
        printf("ERROR flushing i/o buffer on /tyCo/1\n");
        return (ERROR);
    }

    /* open tty device, set appropriate controls and flush both input and
    /* output buffers.
    printf("AVDScomm: opening device /tyCo/2\n");
    if ((ttyb = open ("/tyCo/2", O_RDWR, 0666)) == ERROR) {
        printf("ERROR opening device /tyCo/2\n");
        return (ERROR);
    }

    printf("AVDScomm: setting baud rate on /tyCo/2\n");
    if ((ttyStat = ioctl(ttyb, FIOBAUDRATE, 9600)) == ERROR) {
        printf("ERROR setting baud rate on /tyCo/2\n");
        return (ERROR);
    }

    printf("AVDScomm: setting raw data mode on /tyCo/2\n");
    if ((ttyStat = ioctl(ttyb, FIOSETOPTIONS, OPT_RAW)) == ERROR) {

```

```

        printf("ERROR setting raw data mode on /tyCo/2\n");
        return (ERROR);
    }

    printf("AVDScomm: flushing input buffer on /tyCo/2\n");
    if ((ttyStat = ioctl(ttyb, FIOFLUSH, 0)) == ERROR) {
        printf("ERROR flushing i/o buffer on /tyCo/2\n");
        return (ERROR);
    }

    #ifdef SAVE_OUTPUT_MESSAGE
    if ((fd_out = open("/vxBHome/avds/data/AVDS.outmessage",
        O_CREAT | O_RDWR, 0666)) == ERROR) {
        printf("ERROR opening output message file");
        return (ERROR);
    }
    #endif

    #ifdef SAVE_INPUT_MESSAGE
    if ((fd_in = open("/vxBHome/avds/data/AVDS.inmessage",
        O_CREAT | O_RDWR, 0666)) == ERROR) {
        printf("ERROR opening input message file");
        return (ERROR);
    }
    #endif

    /* initialize global variables
    timeTick = 0;
    communicating = 1;

    /* initialize answer and flight regime
    *(unsigned long *)MEM_ANSWER_POINTER_ADDR = 0xffff; /* no answer
    inMessage.flightRegime = 99; /* set to not valid

    /* user feedback
    printf("AVDScomm: input size = %d\n", sizeof(inMessage));
    printf("AVDScomm: output size = %d\n", sizeof(outMessage));

    /* publish address of the input and output data structures
    printf("AVDScomm: inMessage address: %x\n", &inMessage);
    printf("AVDScomm: outMessage address: %x\n", &outMessage);

    /* create a binary semaphore to provide intertask control
    if ((timesem = semBCreate (SEM_Q_PRIORITY, SEM_EMPTY)) == (SEM_ID)ERROR) {
        printf("ERROR creating semaphore, errno = %d", errno);
        return (ERROR);
    }

    /* spawn the input data task
    if ((inputTask = taskSpawn("AVDSinput", 125, 0, 20000, AVDSinput)) == ERROR) {
        printf("ERROR spawning input task, errno = %d", errno);
        return (ERROR);
    }

    /* setup the signal handler */
    if (signal(SIGALRM, (voidfuncptr) AVDSoutput) == SIG_ERR) {
        printf("ERROR setting signal handler, errno = %d", errno);
        return (ERROR);
    }

    /* create the timer, base timing on real-time clock

```

racoon:/mnt/avds/v5.0/AVDScomm.c

Wed Apr 23 14:33:51 1997

```

if ((tim = timer_create(CLOCK_REALTIME, NULL)) == (timer_t)ERROR) {
    printf("ERROR setting timer, errno = %d", errno);
    return (ERROR);
}
/* initialize timer period and expiration
timectl.it_interval.tv_sec = TIMEOUT_IN_SECS;
timectl.it_interval.tv_nsec = TIMEOUT_IN_NSECS;
timectl.it_value.tv_sec = TIMEOUT_IN_SECS;
timectl.it_value.tv_nsec = TIMEOUT_IN_NSECS;
if (timer_settime(tim, TIMER_ABSTIME, &timectl, NULL) == ERROR) {
    printf("ERROR setting timer, errno = %d", errno);
    return (ERROR);
}

/* keep task alive, but mostly suspended until user flags to stop
do
    sigsuspend(0);
while (communicating);

/* cancel the timer (stop the clocking) */
if (timer_cancel(tim) == ERROR) {
    printf("ERROR deleting timer, errno = %d", errno);
    return (ERROR);
}
/* delete the timer */
if (timer_delete(tim) == ERROR) {
    printf("ERROR deleting timer, errno = %d", errno);
    return (ERROR);
}
/* close the serial port */
close(cttyb);
printf ("AVDScomm exiting \n");
return (OK);
}

/*****
*
* Function: AVDSstop
* OS: VxWorks v5.1.1
* Target: Motorola MVME-166
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: None
* Returns: status
*
* This function is intended to signal end of task, delete all support
* and be ready to re-start.
*
* Rev History: 4/11/95 - Initial design. Errors occur when executes.
* Also doesn't reset properly for an unknown reason.
* This is only a nuisance as the system will start,
* but never need to stop during actual operation. Code
* sent to Wind River for analysis.
*
*****/
STATUS AVDSstop()
{
    /* signal end of task */
    communicating = 0;
}

if (kill(InputTask, SIGALRM) == ERROR) {
    printf("ERROR sending SIGALRM (kill), errno = %d", errno);
    return (ERROR);
}
/* be sure clock runs until wait loop terminates or task will hang */
taskDelay(TIMEOUT_250_mSEC);
/* delete the input task */
if (taskDelete(InputTask) == ERROR) {
    printf("ERROR deleting input task, errno = %d", errno);
    return (ERROR);
}

#define SAVE_OUTPUT_MESSAGE
close (fd_out);
#define
#define SAVE_INPUT_MESSAGE
close (fd_in);
#endif

/*****
*
* Function: calc_checksum
* OS: VxWorks v5.1.1
* Target: Motorola MVME-166
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: buffer - pointer to buffer
* count - size of buffer minus checksum byte
* Returns: checksum
*
* This function calculates a checksum for the data packets sent out.
*
* Rev History: 11/28/95 - Initial design.
*
*****/
unsigned char calc_checksum (unsigned char *buffer, int count)
{
    unsigned char sum;

    sum = 0;
    while (count--)
        sum += *buffer++;
    return ((unsigned char)(0x100 - sum));
}

/*****
*
* Function: checksum
* OS: VxWorks v5.1.1
* Target: Motorola MVME-166
* Author: M. E. Phillips, NRAd, Code 535
* Arguments: buffer - pointer to buffer
* count - size of buffer including checksum byte
* Returns: checksum
*
*****/

```

racoon:/mnt/avds/v5.0/AVDScomm/v2.0/AVDScomm.c

Wed Apr 23 14:33:51 1997

```
* This function verifies a checksum for the data packets received.
*
* Rev History: 11/28/95 - Initial design.
```

```
*****
int checksum (unsigned char *buffer, int count)
{
    unsigned char sum;

    sum = 0;
    while (count-->0)
        sum += *buffer++;
    return (sum);
}
```

```
*****
*
* Function: processAnswer
* OS: VxWorks v5.1.1
* Target: Motorola MVME-166
* Author: M. E. Phillips, NRAoD, Code 535
*
* Arguments: none
* Returns: none
*
* This function reads the answers out of the predetermined VME location
* and places it into the answer queue. If the flight regime counter
* indicates that the regime has been valid for MIN_FR_TICKS, the last
* queue entry (which is effectively MIN_FR_TICKS old) is put into the
* out-going message structure.
*
* Rev History: 12/06/95 - Initial design.
*****
```

```
void processAnswer()
{
    struct MESSAGE_OUT_REC *outRec=&outMessage; /* pointer to output
    #ifdef PRINT_ANSWER
    char printstring[40];
    #endif
```

```
/* put most recent answer into a FIFO queue and update FIFO index to access */
/* the next (last or next out) */
answerQ[answer->ANSWER_QUEUE_SIZE].answer =
(unsigned short) (*(unsigned long *)MEM_ANSWER_POINTER_ADDR);
answerQ[answer->ANSWER_QUEUE_SIZE].hour = inMessage.hour;
answerQ[answer->ANSWER_QUEUE_SIZE].minute = inMessage.minute;
answerQ[answer->ANSWER_QUEUE_SIZE].second = inMessage.second;
answer++;
```

```
/* get Answer from queue if regime conditions are met
if (validFrTicks >= MIN_FR_TICKS) {
    outRec->fault = answerQ[answer->ANSWER_QUEUE_SIZE].answer;
    outRec->status &= ~REGIME_INVALID;
}
else {
    outRec->fault = 0xffff; /* (No Answer) */
}
```

```
    outRec->status |= REGIME_INVALID;
}

/* set output event time to last FIFO queue time */
outRec->eventHour = answerQ[answer->ANSWER_QUEUE_SIZE].hour;
outRec->eventMinute = answerQ[answer->ANSWER_QUEUE_SIZE].minute;
outRec->eventSecond = answerQ[answer->ANSWER_QUEUE_SIZE].second;

#ifdef PRINT_ANSWER
    sprintf(printstring, "\n0x%x, 0x%x, %d\n",
        (short) outRec->fault,
        (short) answerQ[answer->ANSWER_QUEUE_SIZE].answer,
        answer);
    write(tty, printstring, strlen(printstring));
#endif
}
```

racoon:/mnt/avds/v5.0/AVDScomm/v2.0/AVDScomm.c

Wed Apr 23 14:34:06 1997

```
#include <stdio.h>
#include <cacheLib.h>
#include ".../tda/vxPlayback/tda.h"

#define NO_DATA 1

/* temporary control module for lab data acquisition */
/* uses mmvme166 parallel port for Phase Lock Loop */
/* control.

/* NOTE: requires /home/tools/avds/sockets/socket_accept.o module to be */
/* loaded into memory, /home/philipm/vxWorks/PLLControl.

void PLLControl();
void initPort();
int getWatchdog();
void setPort(int PD0, int PDI);
void acceptSocket();
int SocketRead(void *sockBuff, int size);
void SocketClose();
STATUS tda(int command);

/*****
 *
 * output message structure
 *****/
/*****
 *
 * status
 * makes structure an even 14 bytes
 * stored answers, novelty/fault
 * message time
 * message date
 * event time
 *****/

extern int running;
extern struct MESSAGE_OUT_REC *outRec=&outMessage; /* pointer to output */

int PLLstart;
extern int running;
extern struct MESSAGE_OUT_REC *outRec=&outMessage;

/*****
 *
 * Function: PLLControl
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRad, Code 535
 *
 * Arguments: None
 * Returns: status
 *
 * This function is the routine that sets up the socket for the half-duplex
 *****/
```

racoon:/mnt/avds/v5.0/PLLControl/PLLControl.c

Wed Apr 23 14:34:06 1997

```

running = 0;
printf("PLLControl: task complete !!\n");
break;
}
else if (strcmp(sockBuff, "dead") == 0) {
    outRec->status |= NO_DATA;
    printf("PLLControl: A2D not running !!\n");
}
else if (strcmp(sockBuff, "good") == 0) {
    outRec->status &= ~NO_DATA;
    printf("PLLControl: A2D OK !!\n");
}
}
}
SocketClose();
}

/*****
 *
 * Function: initPort
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRAd, Code 535
 * Arguments: None
 * Returns: status
 *
 * This function initializes the parallel port for discrete operation
 *
 * Rev History: 7/27/95 - initial design.
 *****/
void initPort()
{
    short *prCnt0 = (short *)0xffff42030,
          *prCnt1 = (short *)0xffff42032,
          *prCnt2 = (short *)0xffff42034,
          *prCnt3 = (short *)0xffff42036;

    *prCnt0 = 0x0808;
    *prCnt1 = 0x0808;
    *prCnt2 = 0x0800;
    *prCnt3 = 0x0011;
}

/*****
 *
 * Function: getWatchdog
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRAd, Code 535
 * Arguments: None
 * Returns: status
 *
 * This function reads the watchdog signal
 *****/

```

```

 *
 * Rev History: 7/27/95 - initial design.
 *****/

```

```

int getWatchdog ()
{

```

```

    short *prStatAddr = (short *)0xffff42036;
    short prData;

```

```

    prData = *prStatAddr;
    if ((prData & 0x0100) == 0x0100)
        return(1);
    else
        return(0);
}

```

```

/*****
 *
 * Function: setPort
 * OS: VxWorks v5.1.1
 * Target: Motorola MVME-166
 * Author: M. E. Phillips, NRAd, Code 535
 * Arguments: None
 * Returns: status
 *
 * This function sets the state of the gate using the PD0 & PD1 line of the
 * parallel port
 *
 * Rev History: 7/27/95 - initial design.
 *****/

```

```

void setPort(PD0, PD1)
int PD0;
int PD1;
{
    short *prDataAddr = (short *)0xffff4203a;
    *prDataAddr = (short)(PD0 | (PD1<<1));
}

```

Wed Apr 23 14:34:17 1997

```

/*****
 *
 * Program Name:
 *
 * Author: Michael Phillips
 *
 * Module: main routine
 *
 * Function:
 *
 * *****/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define BLK_SIZE 30720

/*****
 *
 * Global Data
 *
 * *****/
int sock, client_sock;
extern int errno;
char dBuff[BLK_SIZE];

/*****
 *
 * Name: acceptSocket
 *
 * Purpose: Accept connections to a named socket as a stream type.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 * *****/
void acceptSocket()
{
    struct sockaddr_in server;
    int port, status;

    port = 2045;
    /* create a socket */
    sock = socket (AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        exit (1);
    }
    fprintf(stderr, "acceptSocket: Error on socket create...\n");
    server.sin_addr.s_addr = INADDR_ANY; /* allow any incoming address */
    server.sin_family = AF_INET;
    server.sin_port = port;
    if (bind (sock, (struct sockaddr *)&server, sizeof (server)) < 0) {
        printf("acceptSocket: bind failed Port %d\n", port);
        exit(1);
    }

    /* allow others to connect to this socket */
    if ((status = listen (sock, 2)) < 0) {
        printf ("acceptSocket: Error in create sockets listen\n");
        exit(1);
    }

    /* wait for a connection */
    printf("acceptSocket: Waiting for connection...\n");
    client_sock = accept (sock, (struct sockaddr *) 0, (int *) 0);

    if (client_sock == -1) {
        printf("acceptSocket: accept failed\n");
        exit(1);
    }
    else printf("acceptSocket: connections accepted ... \n");
}

/*****
 *
 * Name: SocketRead
 *
 * Purpose: reads data sent across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 * *****/
int SocketRead(source, size)
char *source;
int size;
{
    int nbytes;

    nbytes = 0;
    while (nbytes < size) {
        nbytes += read(client_sock, source, size);
        if (nbytes == -1)
            printf("SocketRead: error reading data, errno = %d\n", errno);
    }
    return (nbytes);
}

/*****
 *
 * Name: SocketWrite
 *
 * Purpose: Sends data across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 * *****/
SocketWrite(source, size)
char *source;
int size;
{
    int nbytes;

```

raccoon:/mnt/avds/v5.0/PLUControl/socket_accept.c

Wed Apr 23 14:34:17 1997

```

nbytes = write(client_sock, source, size);
if (nbytes != size)
    if (nbytes == -1)
        printf("SocketWrite: error writing data, errno = %d\n", errno);
    else
        printf("SocketWrite: nbytes not equal to size\n");
}

/*****
 *
 * Name: SocketClose
 *
 * Purpose: Sends data across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
SocketClose()
{
    taskDelay(15);
    printf("SocketClose: close status = %d\n", close(client_sock));
    printf("SocketClose: close status = %d\n", close(sock));
}

/*****
 *
 * Name: wait4Handshake
 *
 * Purpose: provides clocking mechanism between sockets.
 *
 * Rev: 02/03/94 (MEP) - Creation.
 *
 *****/
wait4Handshake()
{
    int nbytes;
    unsigned short sem;

    nbytes = read(client_sock, &sem, sizeof(sem));
    if (nbytes != sizeof(sem))
        if (nbytes == -1)
            printf("wait4Handshake: error reading data, errno = %d\n", errno);
        else
            printf("wait4Handshake: nbytes not equal to size\n");
}

/*****
 *
 * 0 - send next data block
 * 1 - re-send prior data block
 * 2 - abort data transfer
 * 3 - move tape message
 *****/
switch (sem) {
    case 0:
        break;
    case 2:
        close(client_sock);
        exit(0);
}
}

```

Wed Apr 23 14:34:41 1997

```

/*****
 * tda: Tape Data Archiver VxWorks Exabyte SCSI tape utility for AVDS.
 *
 * Author: M. E. Phillips, MRaD, San Diego, CA Mar 95
 * - implemented I/O and shared memory interface
 * - added semaphore control interface
 * - added support for 3 Exabyte 8505XL tape drives
 *****/

#include <stdioLib.h>
#include <ioLib.h>
#include <vxWorks.h>
#include <semLib.h>
#include "exLib.h"
#include "tapeDrv.h"

#include "../sky/include/hostshm.h"
#include "../sky/include/message.h"
#include "tda.h"

/* included facilities */
#define WAIT_FOR_HOST

/* excluded facilities */
#if FALSE
#define DEBUG
#define PRINT_BUFFER_STATUS
#endif

/* In VxWorks, the symbol "_tp0, 1, 2 and _running" must have been added to
the symbol table. */
/* This can be accomplished by executing the following assignment statement: */
VxWorks->_tp0 = (int)0;
VxWorks->_tp1 = (int)0;
VxWorks->_tp2 = (int)0;
VxWorks->_running = (int)0;

extern int tp0;
extern int tp1;
extern int tp2;
extern int running;

int tapeTask;
int memTask;

/* function prototypes */
STATUS exb_close(int *tp);
STATUS exb_delete(char *dev_name);
STATUS exb_open(int *tp, int scsi_unit_num, char *dev_name);
STATUS exb_rewind(int tp);
STATUS exb_locate(int tp, int block);
STATUS exb_ready(int tp);
STATUS exb_where(int tp, int *frame);
STATUS mvBufferToTape();
STATUS mvTapeToBuff();

/*****
 * set system status to OK */
*((int *)MEM_STATUS_POINTER_ADDR) = 0;

switch (command) {
case TDA_REWIND:
    printf("\ntda: EXABYTE REWIND\n\n");
    s = exb_rewind(tp0);
    s = exb_rewind(tp1);
    s = exb_rewind(tp2);
    if (s == ERROR) return(ERROR);
    break;

case TDA_SEEK:
    printf("\ntda: WARNING - Command not yet supported.\n\n");
    break;

case TDA_READ_TAPE:
    printf("\ntda: PLAYBACK mode\n\n");
    printf("\ntda: Installing tape driver\n\n");
    s = tapeDrv();
    if (s == ERROR) {
        printf("exb_open: ERROR - Unable to install Exabyte tape driver.\n\n");
        return(ERROR);
    }
    s = exb_open(&tp0, 4, "rst1");
    if (s == ERROR) return(ERROR);
    s = exb_open(&tp1, 5, "rst2");
    if (s == ERROR) return(ERROR);
    s = exb_open(&tp2, 6, "rst3");
    if (s == ERROR) return(ERROR);
    while (exb_ready(tp0))
        taskDelay(FIVE_SECONDS);
    while (exb_ready(tp1))
        taskDelay(FIVE_SECONDS);
    while (exb_ready(tp2))
        taskDelay(FIVE_SECONDS);

    /* make frame # even multiple if 3 */
    arg /= 3;
    printf("\ntda: tape frame = %d, starting frame = %d\n", arg, arg * 3);
}

```

racocon/mnt/avds/V5.0/tda/vxPlayback/tda.c

Wed Apr 23 14:34:41 1997

```

/* position tape if desired */
if (arg != 0) {
    s = exb_locate (tp0, arg);
    if (s == ERROR) return(ERROR);
    s = exb_locate (tp1, arg);
    if (s == ERROR) return(ERROR);
    s = exb_locate (tp2, arg);
    if (s == ERROR) return(ERROR);
}

s = mvTapeToBuff();
if (s == ERROR) {
    printf("tda: TAPE ERROR or EOF.\n");
    return(ERROR);
}
break;

case TDA_WRITE_TAPE:
    printf("ntda: RECORD mode\n");
    printf("tda: Installing tape driver\n");
    s = tapeDrv();
    if (s == ERROR) {
        printf("exb_open: ERROR - Unable to install Exabyte tape driver.\n");
        return(ERROR);
    }

    s = exb_open(&tp0, 4, "rst1");
    s = exb_open(&tp1, 5, "rst2");
    s = exb_open(&tp2, 6, "rst3");
    if (s == ERROR) return(ERROR);
    running = 0;
    s = mvBuffToTape();
    if (s == ERROR) return(ERROR);
    break;

case TDA_STOP:
    printf("ntda: STOP\n");
    running = 0;
    taskDelay(ONE_SECOND);
    s = exb_rewind(tp0);
    s = exb_rewind(tp1);
    s = exb_rewind(tp2);
    if (s == ERROR) return(ERROR);
    s = exb_close(&tp0);
    s = exb_close(&tp1);
    s = exb_close(&tp2);
    if (s == ERROR) return(ERROR);
    s = exb_delete("rst1");
    s = exb_delete("rst2");
    s = exb_delete("rst3");
    if (s == ERROR) return(ERROR);

    printf("exb_close: Removing tape driver\n");
    s = tapeDrvRemove(TRUE);
    if (s == ERROR) {
        printf("exb_close: ERROR - Exabyte tape driver was not found.\n");
        return(ERROR);
    }

    break;

default:
    printf("tda: ERROR - Illegal TDA command %d.\n", command);
    return(ERROR);
} /* END switch (command) */

taskDelay(ONE_SECOND);
return(OK);
} /* END tda() */

/*****
STATUS exb_close (tp)
int *tp;
{
    STATUS s;

    printf("exb_close: closing tape\n");
    s = close(*tp);
    if (s == ERROR) {
        printf("exb_close: ERROR - %d is invalid during close.\n", *tp);
        return(ERROR);
    }
    *tp = 0;
}

/*****
STATUS exb_delete (dev_name)
char *dev_name;
{
    STATUS s;

    printf("exb_close: Deleting tape device\n");
    s = tapeDevDelete(dev_name);
    if (s == ERROR) {
        printf("exb_close: ERROR - Exabyte tape device was not found.\n");
        return(ERROR);
    }

    return(OK);
} /* END exb_close() */

/*****
STATUS exb_open (tp, scsi_unit_num, dev_name)
int *tp;
int scsi_unit_num;
char *dev_name;
{
    STATUS s;

    printf("exb_open: Creating tape device\n");
    s = tapeDevCreate(dev_name, scsi_unit_num, EXB_BUFF_SIZE);
    if (s == ERROR) {
        s = tapeDevDelete(dev_name);
        if (s == ERROR) {

```

raccoon:/mnt/avds/v5.0/tda/vxPlayback/tda.c

Wed Apr 23 14:34:41 1997

```

    )
    s = tapeDevCreate(dev_name, scsi_unit_num, EXB_BUFF_SIZE);
    if (s == ERROR) {
        printf("exb_open: ERROR - Unable to add Exabyte tape device.\n");
        return(ERROR);
    }
}

/* dummy open to get past first time "Unit Attention End-of-medium" error */
exbOpen(scsi_unit_num, EXB_MODE, EXB_BUFF_SIZE);

printf("exb_open: opening tape device\n");
*tp = open(dev_name, O_RDWR, EXB_MODE);
if (*tp == -1) {
    printf("exb_open: ERROR - Unable to open Exabyte tape device.\n");
    return(ERROR);
}

return(OK);
} /* END exb_open() */

/*****
STATUS exb_rewind (tp)
int tp;
{
    STATUS s;

    printf("exb_rewind: Sending rewind command.\n");
    taskDelay(FIVE_SECONDS);
    s = ioctl(tp, TIOWIND, 0);
    if (s == ERROR) {
        printf("exb_rewind: ERROR - Problem rewinding Exabyte tape.\n");
        return(ERROR);
    }
    printf("exb_rewind: Rewind command sent\n");
    return(OK);
} /* END exb_rewind() */

/*****
STATUS mvBuffToTape ()
{
    STATUS s;
    unsigned long *BCB_Write;
    unsigned long *BCB_Read;
    unsigned long BCB_W;
    unsigned long BCB_R;
    register unsigned char *vmeAddr0;
    register unsigned char *vmeAddr1;
    register unsigned char *vmeAddr2;
    unsigned char localBuff[BLOCK_SIZE];
    int block;
    int nb;
    int tapeFrame = 0;
    int frameToProcess = 0;
    int waiting = 1;

    printf("mvBuffToTape: executing.\n");
    BCB_Write = (unsigned long *)MEM_WRITE_POINTER_ADDR;

```

racoona/mnt/avds/v5.0/ida/vxPlayback/ida.c

Wed Apr 23 14:34:41 1997

```

BCB_Read = (unsigned long *)MEM_READ_POINTER_ADDR;
printf("mvBuffToTape: Waiting for start!!\n");
while(!running);
printf("mvBuffToTape: Started.\n");

while (running) {
    /* get data buffer from memory board to local memory */
    BCB_W = *BCB_Write;
    BCB_R = *BCB_Read;
    framesToProcess = (BCB_W - BCB_R) / BUFFER_SIZE;
    /* adjust the frames for wrap around case (BCB_W < BCB_R) */
    if (framesToProcess < 0) framesToProcess += NUM_BUFFERS;

    #ifdef DEBUG
        printf("mvBuffToTape: BCB_W = %x, BCB_R = %x, Frames = %d\n",
            BCB_W, BCB_R, framesToProcess);
    #endif

    if (framesToProcess >= 3) {
        /* data available, read buffer */
        vmeAddr0 = (unsigned char *) (MEM_BASE_ADDR + BCB_R);
        vmeAddr1 = (unsigned char *) (MEM_BASE_ADDR + BCB_R);
        vmeAddr2 = (unsigned char *) (MEM_BASE_ADDR + BCB_R + BUFFER_SIZE);
        block = BLOCKS_PER_BUFFER;
        while(block--) {
            bcopy (vmeAddr0, localBuff, BLOCK_SIZE);
            nb = write(tp0, localBuff, BLOCK_SIZE);
            if (nb != BLOCK_SIZE) {
                printf("mvBuffToTape: Error writing to tape #0.\n");
                return(ERROR);
            }
            bcopy (vmeAddr1, localBuff, BLOCK_SIZE);
            nb = write(tp1, localBuff, BLOCK_SIZE);
            if (nb != BLOCK_SIZE) {
                printf("mvBuffToTape: Error writing to tape #1.\n");
                return(ERROR);
            }
            bcopy (vmeAddr2, localBuff, BLOCK_SIZE);
            nb = write(tp2, localBuff, BLOCK_SIZE);
            if (nb != BLOCK_SIZE) {
                printf("mvBuffToTape: Error writing to tape #2.\n");
                return(ERROR);
            }
            vmeAddr0 += BLOCK_SIZE;
            vmeAddr1 += BLOCK_SIZE;
            vmeAddr2 += BLOCK_SIZE;
        }

        /* update the frame number to the tape */
        tapeFrame += 3;

        /* update BCB */
        BCB_R += (BUFFER_SIZE*3);
        if (BCB_R == BUFFER_END_ADDR)
            *BCB_Read = 0;
        else
            *BCB_Read = BCB_R;

        if (!waiting) {
            waiting = 0;
            printf("\nrunning!\n");
        }
        else {
            /* no data available, wait 0.4 seconds before checking again */
            printf("Waiting for data !!\n");
            taskDelay(BUFF_DELAY);
            waiting = 1;
        }
    }
    printf("mvBuffToTape: stop command received.\n");
    /* write filemark to tape */
    printf("mvBuffToTape: writing end-of-filemark.\n");
    s = ioctl(tp0, TIOWRITEFMK, 1);
    if (s == ERROR) {
        printf("mvLocalBuffToTape: Error writing filemark to tape.\n");
        return(ERROR);
    }
    s = ioctl(tp1, TIOWRITEFMK, 1);
    if (s == ERROR) {
        printf("mvLocalBuffToTape: Error writing filemark to tape.\n");
        return(ERROR);
    }
    s = ioctl(tp2, TIOWRITEFMK, 1);
    if (s == ERROR) {
        printf("mvLocalBuffToTape: Error writing filemark to tape.\n");
        return(ERROR);
    }
    printf("mvBuffToTape: exiting.\n");
    return(OK);
}

/***** mvTapeToBuff ( ) *****/
STATUS mvTapeToBuff (
    STATUS s;
    unsigned long *BCB_Write;
    unsigned long *BCB_Read;
    unsigned long BCB_W;
    unsigned long BCB_R;
    register unsigned char *vmeAddr0;
    register unsigned char *vmeAddr1;
    register unsigned char *vmeAddr2;
    unsigned char localBuff[BLOCK_SIZE];
    int nb;
    int tapeFrame = 0;
) {
    printf("mvTapeToBuff: executing.\n");

    BCB_Write = (unsigned long *)MEM_WRITE_POINTER_ADDR;
    BCB_Read = (unsigned long *)MEM_READ_POINTER_ADDR;
    *BCB_Write = 0;
    *BCB_Read = 0;

    printf("mvTapeToBuff: Waiting for start!!\n");
    while(!running);
    printf("mvTapeToBuff: Started.\n");
    while (running) {
        /* get data buffer from memory board to local memory */

```

racocon:/mnt/avds/v5.0/ida/vxPlayback/ida.c

Wed Apr 23 14:34:41 1997

```

BCB_W = *BCB_Write;
BCB_R = *BCB_Read;

#ifdef DEBUG
printf("mvTapeToBuff: CCB_W = %x, CCB_R = %x, Frame = %d\n",
      CCB_W, CCB_R, tapeFrame);
#endif

/* data available, read buffer */
vmeAddr0 = (unsigned char *) (MEM_BASE_ADDR + CCB_W);
vmeAddr1 = (unsigned char *) (MEM_BASE_ADDR + CCB_W + BUFFER_SIZE);
vmeAddr2 = (unsigned char *) (MEM_BASE_ADDR + CCB_W + (BUFFER_SIZE*2));
block = BLOCKS_PER_BUFFER;
while (block--) {
    nb = read(tp0, localBuff, BLOCK_SIZE);
    if (nb != BLOCK_SIZE) {
        printf("mvTapeToBuff: Error reading to tape #0, nb= %d.\n", nb);
        *((int *) MEM_STATUS_POINTER_ADDR) = -1;
        return(ERROR);
    }
    bcopy (localBuff, vmeAddr0, BLOCK_SIZE);
    nb = read(tp1, localBuff, BLOCK_SIZE);
    if (nb != BLOCK_SIZE) {
        printf("mvTapeToBuff: Error reading to tape #1, nb= %d.\n", nb);
        *((int *) MEM_STATUS_POINTER_ADDR) = -1;
        return(ERROR);
    }
    bcopy (localBuff, vmeAddr1, BLOCK_SIZE);
    nb = read(tp2, localBuff, BLOCK_SIZE);
    if (nb != BLOCK_SIZE) {
        printf("mvTapeToBuff: Error reading to tape #2, nb= %d.\n", nb);
        *((int *) MEM_STATUS_POINTER_ADDR) = -1;
        return(ERROR);
    }
    bcopy (localBuff, vmeAddr2, BLOCK_SIZE);
    vmeAddr0 += BLOCK_SIZE;
    vmeAddr1 += BLOCK_SIZE;
    vmeAddr2 += BLOCK_SIZE;
}

/* update the frame number to the tape */
tapeFrame += 3;

/* update CCB */
CCB_W += (BUFFER_SIZE*3);
if (CCB_W == BUFFER_END_ADDR) {
#ifdef WAIT_FOR_HOST
    /* wait for the data collector to catch up */
    do {
        printf("Waiting for host\n");
        taskDelay (TEN_SECONDS);
        /* get data buffer from memory board to local memory */
        CCB_W = *CCB_Write;
        CCB_R = *CCB_Read;
    } while ((CCB_W != CCB_R) && running);
#endif
    *CCB_Write = 0;
}
else
    *CCB_Write = CCB_W;
}

```

raccoon/mnt/avds/v5.0/ida/vxPlayback/ida.c

Wed Apr 23 14:34:45 1997

```

/* addresses and control for memory storage */
#define MEM_BASE_ADDR 0x60000000
#define MEM_SIZE 0x08000000
#define MEM_ANSWER_POINTER_ADDR (MEM_BASE_ADDR + MEM_SIZE - 16)
#define MEM_STATUS_POINTER_ADDR (MEM_BASE_ADDR + MEM_SIZE - 12)
#define MEM_WRITE_POINTER_ADDR (MEM_BASE_ADDR + MEM_SIZE - 8)
#define MEM_READ_POINTER_ADDR (MEM_BASE_ADDR + MEM_SIZE - 4)
#define MEM_COMM_POINTER_ADDR (MEM_ANSWER_POINTER_ADDR - (sizeof(inMessage) +
sizeof(outMessage)))

#define BUFFER_SIZE sizeof(tData)
#define LONG_BUFFER_SIZE BUFFER_SIZE/4
#define TOTAL_BUFFERS (int) (MEM_SIZE / BUFFER_SIZE)
#define NUM_BUFFERS (TOTAL_BUFFERS - TOTAL_BUFFERS*3)
#define INITIAL_BUFFER_OFFSET 0
#define BUFFER_END_ADDR (BUFFER_SIZE * NUM_BUFFERS)
#define BLOCK_SIZE 0x4000
#define BLOCKS_PER_BUFFER BUFFER_SIZE / BLOCK_SIZE
#define BUFF_A 0
#define BUFF_B 1

/* exabyte driver constants */
#define EXB_BUFF_SIZE BLOCK_SIZE
#define EXB_MODE 0x644

/* TDA COMMANDS */
#define TDA_AUTO_SENSE 21
#define TDA_PARMS 22
#define TDA_REWIND 23
#define TDA_SEEK 24
#define TDA_STOP 26
#define TDA_WRITE_TAPE 200
#define TDA_READ_TAPE 201

/* delay constants */
#define TICK 1 /* 16.67 ms per tick */
#define ONE_SECOND 60 /* ticks */
#define FIVE_SECONDS 300 /* ticks */
#define TEN_SECONDS 600 /* ticks */
#define BUFF_DELAY 24
#define WAIT_FOR_TAPE ONE_SECOND
/* usage example: taskDelay(ONE_SECOND); */

```

racoona/mnt/avds/v5.0/tda/vxPlayback/tda.h

Wed Apr 23 14:34:57 1997

```

printf("error writing data, errno = %d\n", errno);
else
    printf("nbytes not equal to size\n");
}

/*****
 *
 * Name: main
 *
 * Purpose: tells server to send next block.
 *
 * Rev: 02/03/94 (MEP) - Creation.
 *
 *****/
main(argc, argv) /* test version */
int argc;
char **argv;
{
    int size, numBlocks, nb;
    char ans, dummy;
    int command;
    int stat;

    if (argc < 2) {
        printf("USAGE: connect <servername>\n\n");
        exit(0);
    }
    createConnectionSocket(argv[1]);

    while (ans != 'q') {
        printf ("Enter code: ");
        ans = getchar();

        if (ans != 'q')
            if (strcmp("0123456", ans) != NULL) {
                command = (int) (ans - 48);
                write(sock, &command, sizeof(command));
                printf("command = %d\n", command);
            }
            else
                printf ("Enter a correct command! \n");

        dummy = getchar(); /* clear buffer */
    }

    printf("closing connection\n");
    command = 999;
    write(sock, &command, sizeof(command));
    stat = close (sock);
}

```

Wed Apr 23 14:34:57 1997

```

/*****
 *
 * Program Name:
 *
 * Author: Michael Phillips
 *
 * Module: main routine
 *
 * Function:
 *
 *****/
 *
 * Include Files:
 *
 * -----
 *
 * #include <stdio.h>
 * #include <sys/types.h>
 * #include <sys/socket.h>
 * #include <netinet/in.h>
 * #include <netdb.h>
 *
 * #define SOCKET_OPEN_TRIES 40
 *
 *****/
 *
 * Global Data
 *
 * -----
 *
 */
 *
 * int sock;
 * extern errno;
 *
 *****/
 *
 * Name: createSockets
 *
 * Purpose: Connect to a named socket as a stream type.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
 *
 * createConnectionSocket(servername)
 * char *servername;
 *
 {
     struct sockaddr_in server;
     struct hostent *hp, *gethostbyname();
     int status, port;
     int errCnt=SOCKET_OPEN_TRIES;

     port = 2046;
     status = -1;
     server.sin_port = port;
     server.sin_family = AF_INET;
     hp = gethostbyname(servername);
     if (hp == 0) {
         fprintf(stderr, "unknown host\n");
         exit(2);
     }
     bcopy((char *)hp->h_addr, (char *)&server.sin_addr, hp->h_length);

```

```

     printf("waiting for a connection...\n");
     fflush(stdout);
     while((status == -1) && (errCnt--)) {
         sock = socket(AF_INET, SOCK_STREAM, 0);
         status = connect(sock, (struct sockaddr *)&server, sizeof(server));
         if (status == -1) {
             close(sock);
             sleep(1);
         }
     }
     if (status < 0) {
         printf("error making connection\n");
         exit(-1);
     }
     else
         printf("connection established.\n");
 }
/*****
 *
 * Name: readSocketData
 *
 * Purpose: reads data sent across a socket connection.
 *
 * Rev: 02/02/94 (MEP) - Creation.
 *
 *****/
 *
 * int readSocketData(source, size)
 * char *source;
 * int size;
 *
 {
     int nbytes;

     nbytes = 0;
     while (nbytes < size) {
         nbytes += read(sock, source, size);
         if (nbytes == -1)
             printf("error reading data, errno = %d\n", errno);
     }
     return (nbytes);
 }
/*****
 *
 * Name: sendHandshake
 *
 * Purpose: tells server to send next block.
 *
 * Rev: 02/03/94 (MEP) - Creation.
 *
 *****/
 *
 * sendHandshake()
 *
 {
     int nbytes;
     unsigned short go = 0;

     nbytes = write(sock, &go, sizeof(go));
     if (nbytes != sizeof(go))
         if (nbytes == -1)

```

racocon:/mnt/avds/v5.0/tda/vxPlayback/socket_connect.c

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 1997		3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE AIR VEHICLE DIAGNOSTIC SYSTEM: CH-46 AFT MAIN TRANS- MISSION FAULT DIAGNOSIS—FINAL REPORT				5. FUNDING NUMBERS PE: 0603792N	
6. AUTHOR(S) K. G. Church, R. R. Kolesar, M. E. Phillips, R. C. Garrido					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division (NRaD) San Diego, California 92152-5001				8. PERFORMING ORGANIZATION REPORT NUMBER TD 2966	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Chief of Naval Operations Washington, DC 20350-2000				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The goal of the helicopter drive system condition monitoring component of the Air Vehicle Diagnostic System (AVDS) program is to develop technology that will facilitate transition to "condition-based maintenance" (CBM) for these components. The specific system requirements as stated in the AVDS Execution Plan are as follows: • Detect and classify gearbox faults in real time in flight. • Cope with variations between gearboxes. • Perform diagnostics in a variety of flight regimes. • Reduce false-alarm rates. • Reduce maintenance costs and facilitate transition to CBM. This report describes the performance of a system developed by NRaD that meets these requirements.					
14. SUBJECT TERMS Mission area: Diagnostics and Maintenance false-alarm rates helicopter gearbox maintenance diagnostics condition-based maintenance (CBM)				15. NUMBER OF PAGES 130	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT		

21a. NAME OF RESPONSIBLE INDIVIDUAL

R. R. Kolesar

21b. TELEPHONE (include Area Code)

(619) 553-9893

email: kolesar@nosc.mil

21c. OFFICE SYMBOL

Code D374

INITIAL DISTRIBUTION

Code D0012	Patent Counsel	(1)
Code D0271	Archive/Stock	(2)
Code D0274	Library	(2)
Code D0271	D. Richter	(1)
Code D374	R. R. Kolesar	(26)

Defense Technical Information Center
Fort Belvoir, VA 22060-6218 (4)

NCCOSC Washington Liaison Office
Arlington, VA 22245-5200

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Arlington, VA 22244-5114

GIDEP Operations Center
Corona, CA 91718-8000