NASA Conference Publication 2206

Ruggedized Minicomputer Hardware and Software Topics - 1981

Proceedings of a conference held in
San Diego, California
February 22-25, 1981

NASA

NASA Conference Publication 2206

Ruggedized Minicomputer Hardware and Software Topics - 1981

Proceedings of a conference held in San Diego, California February 22-25, 1981

NASA
National Aeronautics and Space Administration

Scientific and Technical Information Branch

1981

PREFACE

This conference publication contains the proceedings of the Fourth ROLM MIL-SPEC Computer Users Group Conference, held in San Diego, California, February 22-25, 1981. The main purposes of the conference were (1) to promote the interchange of ideas among users of ruggedized minicomputers through description of individual applications, and (2) to report to the computer manufacturer any concerns individual users had relating to the operation of either hardware or software supplied by the manufacturer. While all conference activities related to the use of a single manufacturer's ruggedized computers, many of the novel ideas discussed at this conference have a much wider scope of applicability. None of the company/user interchanges relating to the use of ruggedized minicomputers manufactured by a specific vendor has been included in this publication.

The Fourth Users Group Conference contained presentations covering a wide range of topics, including (1) the role of minicomputers in the development and/or certification of new commercial or military airplanes in both the United States and Europe, (2) generalized software error detection techniques, (3) real-time software development tools, (4) a redundancy management research tool for aircraft navigation/flight control sensors, (5) extended memory management techniques using a high-order language, and (6) some comments on establishing a system maintenance scheme. In addition, copies of the slides used by the guest speaker detailing areas of new U.S. Navy research and development efforts for 1982 have been included.

The use of trade names or names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

Wayne H. Bryant
Eastern Area Vice-President
ROLM MIL-SPEC Computer Users Group

CONTENTS

AIRBORNE DATA ANALYSIS/MONITOR SYSTEM

Darryl B. Stephison
Boeing Commercial Airplane Company
Seattle, Washington

ABSTRACT

The Airborne Data Analysis/Monitor System (ADAMS) is a ROLM 1666 computer-based system installed onboard test airplanes and used during experimental testing of those airplanes. ADAMS provides real-time displays to enable onboard test engineers to make rapid decisions about the conduct of the test. Such decisions have reduced the cost and the time required to certify new model airplanes. These decisions have also improved the quality of data derived from the test, leading to more rapid development of improvements resulting in quieter, safer, and more efficient airplanes. The availability of airborne data processing removes most of the weather and geographical restrictions imposed by telemetered flight test data systems.

ADAMS receives sensory input from a separate onboard data acquisition and recording system. Sensory data is converted to engineering units using automatically selected transform functions matching the characteristics of the data acquisition system. Depending on operator selected options, a variety of more complex data transformations are performed to reduce the large volume of data to more meaningful indicators of data quality, test conduct, and airplane performance. The operator may also select several output devices and/or formats to meet the needs of the particular test. A data base is maintained to describe the airplane, the data acquisition system, the type of testing, and the conditions under which the test is being performed.

In addition to the 1666 computer, the ADAMS hardware includes a DDC System 90 fixed head disk and a Miltope DD400 floppy disk. Boeing has designed a DMA interface to the data acquisition system and an intelligent terminal to reduce system overhead and simplify operator commands. The ADAMS software includes RMX/RTOS and both ROLM FORTRAN and assembly language are used.

1.0 INTRODUCTION

The Boeing Commercial Airplane Company is currently unchallenged as the nation's leader in commercial aircraft sales. The company's ability to design, build, and market better airplanes and a greater variety of airplanes is significantly dependent on the ability to prove the airworthiness of those airplanes and to provide data for continuing product improvement. The flight testing of commercial jet airplanes to serve those needs has always been expensive and time consuming. The post-test analysis of flight test data frequently showed that tests had not been performed correctly or that target parameter values had not been reached. This resulted in repeat testing. Boeing is now using onboard digital computer systems for analysis of flight test data in real time. This enables engineers onboard the test aircraft to make rapid decisions about the conduct of the test. Such decisions have reduced the cost and the time required to certify new model airplanes and have improved the quality of data derived from the test. The availability of airborne data processing removes most of the weather and geographical restrictions imposed by telemetered flight test data systems.

1.1 Background

Boeing first used computer data processing to aid in the analysis of flight test data in the early 1950's as part of the B-52 flight test program. Manual calculations were unsatisfactory, especially for determination of net thrust for an eight-engine airplane. With the introduction of commercial jet transports, Boeing began to record flight test data on magnetic tape. These magnetic tapes were used as input to ground-based computer data processing systems for post test data analysis. Throughout the 707, 727, and 737 projects, improvements were made to the data acquisition, data recording, and data processing systems. Computing techniques were continually developed, progressing from the IBM 701 through the UNIVAC 1103A, the IBM 7094, and the IBM 360.

The concept of an airborne data monitor based on a general purpose minicomputer was introduced in the early 1970's in connection with the Pulse Code Modulation (PCM) data acquisition system to be used in testing of the E3A (AWACS) airplane. The AWACS Preflight and Data Acquisition System (APDAS) was implemented in 1973. This system was based on a Data General NOVA 1220 computer. When a similar system was proposed for commercial airplane testing, the need for a more rugged main frame was identified. Experience with APDAS had shown that more processing could be done in this type of system and the concept of "Application Programs" was introduced. The Airborne Data Analysis-Monitor System (ADAMS) was implemented in 1975 to provide an onboard, real time, data monitor and analysis capability based on the ROLM 1602 Ruggednova.

In 1977, after an extensive review of ADAMS capabilities, problems, and deficiencies, a decision was made to redevelop ADAMS using the most up-to-date design and implementation methodologies practical for the project. After a considerable effort to define and document the requirements for the system and a study of hardware and software tools available, a design concept was adopted based on the ROLM 1666 Processor, the DDC System 90 Fixed Head Disk, RMX/RDOS, and ROLM FORTRAN. About the same time a decision was also made to redevelop the ground-based computing system using the IBM 303X Processor and several Digital Equipment Corporation PDP-11 Processors as peripheral processors. The thrust of these redevelopment efforts was the impending 767 and 757 airplane certification test programs scheduled to begin in October of 1981.

A prototype system was installed on an airplane in early 1980 to make an initial test of the system. The system was known to have several bugs and only a subset of the total functions to be included in the final system. While this was expected to prompt many complaints, the single item most vividly identified in this demonstration was the painfully slow system response.

Simple instrumentation (to be presented in another paper) was very helpful in locating the source of our sluggish performance. As much as 80% of all instruction executions were in RMX/RDOS system space. As a result of further investigation, a decision was made to change operating systems in midstream to RMX/RTOS. This involved a considerable conversion effort which was not in the project plan. By late summer, the conversion had been successfully completed. Additional optimizing of RMX/RTOS and our own executive subsystem have brought a reasonable performance level into view.

1.2 Environment

ADAMS is part of a complex data gathering and processing capability as shown in Figure 1. Onboard the test airplane, inputs from transducers and electrical/electronic systems in the airplane are combined into serial pulse code modulated bit streams by the Data Acquisition System. The serial bit streams are recorded on magnetic tape and/or input to ADAMS. The magnetic tape is used after the flight as input to the Data Processing Ground Station, the purpose of which is to strip selected parameters from the magnetic tape and pass these to the Test Data Processing System for final processing. Final data is output as graphical displays, tabulations, hardcopy plots, and magnetic tape files. Much support is required in terms of data base parameters to keep all processes running. A large data base is maintained by a Data Base Management System. Data base parameters required to support the airborne systems are written to the appropriate storage media by the Instrumentation Sub-System. This data consists of PROM's to support the Data Acquisition System and floppy disk files for ADAMS.

ADAMS thus has four major external interfaces as shown in Figure 2. The primary data flow into the system is the serial PCM code received from the Data Acquisition System. The primary output from the system is display information. The system is transaction driven with most processing being initiated by commands from the operator. Data base files and program files are received from the Instrumentation Sub-System in the form of RMX/RDOS compatible floppy disk files.

2.0 HARDWARE

The hardware for ADAMS can be simply viewed as a central processor surrounded by four major types of peripheral equipment corresponding to the four major external interfaces as shown in Figure 3.

2.1 Central Processor

The ADAMS II Central Processor is a ROLM 1666 computer including the main frame, control panel, and 16-slot I/O expansion chassis. The I/O expansion chassis contains an I/O Bus Repeater, a Disk Controller, an Asynchronous Line Multiplexer, a Basic I/O Interface, and a Floppy Disk Interface purchased from ROLM. In addition to these purchased interfaces, Boeing has designed and built an interface for the Datametrics DMC1500 Line Printer and a Measurement Data Bus Interface which is part of our PCM subsystem.

2.2 PCM Input

The PCM Input Subsystem is shown in Figure 4. This subsystem converts serial PCM data to parallel data, identifies each word of data and places each data word and its identification on the Measurement Data Bus. Each data word is then transferred to the core memory of the ROLM 1666 by direct memory access. The PCM Decommutator provides bit, frame, and subframe synchronization and converts the serial PCM to parallel data. The Word Identifier provides a unique identification for each word of parallel data. Boeing has designed and built the MDB Interface which deposits the data in core memory based on the identification.

2.3 Display Devices

The major Display Devices are shown in Figure 5. There are two operator stations, each consisting of a CRT, keyboard, and terminal controller. Boeing repackaged a commercial CRT to make it suitable for airborne service. The Boeing-designed terminal controller includes an Intel SBC 80/204 single board computer. Typical terminal functions are enhanced by the addition of control functions unique to ADAMS. These functions relieve the Central Processor of overhead processing required for a more generalized terminal. A graphics display capability consisting of a Graphics Controller, CRT and a Plotter is currently being integrated into the system. Analog output from the system is available from a Digital-to-Analog Converter and a Measurement Selector. Connection of these devices to the Measurement Data Bus enables the system to output either selected raw PCM parameters or processed data from the ROLM 1666. The Line Printer gives alphanumeric hardcopy output capability. The Remote Digital Display is a Boeing-designed, five-digit numeric indicator driven by RS232C signals from an ALM port.

2.4 Disk

The mass storage capability of ADAMS consists of a fixed head disk and a floppy disk. The 4M byte fixed head disk has been modified for airborne use by the substitution of a 400 Hz motor and power supply. The two-drive floppy disk is mainly used for transportation of data base and program files from ground based systems to the airplane in preparation for a test flight. During normal operation of ADAMS, only the fixed head disk is accessed.

2.5 Keyboard

The simplest of hardware components in ADAMS is the keyboard. This is a Microswitch Keyboard which has been packaged by Boeing to make it suitable for the airborne environment. The keyboard is connected to the Terminal Controller. In addition to typical keyboard functions, the ADAMS keyboard has a number of fixed string keys and eight user defined string keys. The Terminal Controller handles character echoing and other command processing functions to relieve Central Processor overhead.

3.0 SOFTWARE

The software within the Central Processor can be viewed as one or more Applications surrounded by peripheral processes as shown in Figure 6. As at the hardware level, each of the peripheral process types is related to one of the four major external interfaces. Measurement processing converts the raw PCM parameters to engineering units parameters (i.e. having units such as degrees, pounds, volts, etc.). Display processing includes device drivers and additional routines to allocate devices to Applications or resolve conflicts when two or more Applications compete for use of a Display Device. Transaction processing accepts operator commands, loads Application program code, schedules Application execution, and passes command arguments to the Applications. File Processing includes disk I/O drivers, a form of data base management, and temporary disk file management.

The peripheral processes shown in Figure 6 have been divided into Operating System and Executive functions. The portion of device handling, memory

management, and task management handled by the ROLM RMX/RTOS are called Operating System functions. Additional resource management, measurement processing, transaction processing, file management, data base management, and system initialization functions unique to ADAMS are called Executive. There are three types of Application functions as shown in Figure 7. These are called Monitor, Data Analysis, and Utility functions. Additional "stand-alone" software is also provided with ADAMS. Thus, the breakdown of ADAMS software is shown in Figure 8.

3.1 Operating System

The ADAMS Operating System is basically the ROLM RMX/RTOS. The system has been modified by Boeing with the substitution of an improved terminal driver, an improved ALM driver, an improved line printer driver, an improved real time clock handler, an improved power fail recovery, an improved system error handler, and the addition of a Measurement Data Bus driver. A file manager has been added to allow creation, deletion, opening, reading, and writing of temporary disk files.

3.2 Executive

The Executive performs supervisory functions necessary to support execution of Monitor, Data Analysis and Utility functions performed by ADAMS. The supervisory functions are in areas of measurement processing, device management and allocation, data base management, transaction processing, system initialization, and other miscellaneous processing.

A more detailed discussion of the ADAMS Executive will be presented in another paper.

3.3 Monitor

The ADAMS Monitor functions are the primary display generating functions. These functions are used to generate standard displays of either PCM parameters or computed parameters.

The Quicklook function selectively builds displays of the current engineering units or raw PCM for up to 20 measurements. The update rate of this display is approximately once per second. Measurement identification number, title, and units are also displayed. Measurement values are evaluated against predefined preflight or flight limit values. Displayed measurements exceeding these limits are visibly identified.

The Hardcopy function controls the transfer of data from the operator display screen to the printer in response to a command. No tranformation of data occurs.

The Printer Time History function generates a display on the printer of a tabulated listing of engineering units data for up to ten measurements. Selectable sample rates from one sample in ten seconds to ten samples per second are provided with default to one sample per second. Measurement identification number and units are provided at the beginning of each tabulation. Operator event marking of the output is also provided.

The Analog function selects, scales, and outputs up to 16 engineering units parameters to the digital to analog converter.

The Panel function selects up to 20 engineering units parameters and outputs these parameters to panel display modules. A positive indication is made to the panel display observer if and when the Panel function is not updating. Operator selected panel update rates of from one to five samples per second are provided with default to one sample per second.

The Graphics function selects up to 20 engineering units parameters plus time and formats these for output using the graphics display. Up to seven parameters are displayed in real-time. The remaining parameters are stored for non-real-time plotting. Storage is sufficient for up to 10,000 data values. Plot formats are of two types, X-Y plot or strip chart. Format information is entered manually or on file records created using FTCS. In the strip chart format up to six engineering units parameters are displayed as a function of time in a manner similar to the chart recorder analog output. In this format the storage of the Graphics function is used to store old data which has been "scrolled" off from the display. This old data may be later redisplayed as a non-real-time plot.

3.4 Data Analysis

The Data Analysis functions convert PCM parameters to computed parameters. In addition, these functions may build special displays. The first three of these functions are fundamental and are prerequisites for operation of several other Data Analysis functions.

The Gross Weight function computes current gross weight of the airplane and fuel density for each engine.

The Basic Airplane function computes various fundamental aerodynamic parameters such as airspeed, altitude, Mach number, ambient air temperature, and lift coefficient.

The Engine Thrust function computes engine net thrust and various other engine performance parameters using generalized engine thrust curves.

The General Calculations function enables the user of ADAMS to define real-time processing to be done by the system with a minimum of design flow time. Definitions are entered in the form of FORTRAN assignment statements.

The Averages function enables the user of ADAMS to define summary processing to be done by the system with a minimum of design flow time. Definitions include averages, minimum, maximum, slopes and integrals.

The Cruise function computes various periodic parameters relevant to cruise performance testing. These parameters are made available for display using the various monitor functions and also stored for later post-condition processing by the Cruise Summary function.

The Cruise Summary program computes slopes and averages for various parameters stored by the Cruise function and uses the results to refine the computations of the Cruise function.

The Take-Off function computes various real-time and summary parameters relevant to take-off and landing performance testing.

The Stalls function computes various real-time and summary parameters relevant to stall performance testing.

The Flight Controls function computes various real-time parameters relevant to stability and control testing.

The Acoustics function computes various real time and summary parameters relevant to aircraft noise testing. Summary parameters are displayed on the screen. The operator is able to "edit" the summary data. The Acoustics function then transmits the summary data to an acoustics data processing system on the ground.

The Loads function is used during structural testing to combine several PCM parameters according to "linear multiple equations" to produce real-time parameters. In addition, this function compares PCM parameters to pre-defined limits and produces reports of discrepancies.

The Power Plant function computes several real-time and summary parameters relevant to engine evaluation testing.

The Winds function is used during cruise performance testing to determine wind speed and direction and other real-time parameters useful in analyzing cruise data.

The Deviations function is used to compute the deviation of an input parameter from a steady state value.

The Rosette function is used in airplane structural testing to compute total stress from rosette strain gages.

The Pressure Coefficients function is used in pressure survey testing to compute coefficients of port pressures (ratio of port pressure to reference pressure) and to plot pressure distribution as a function of port position both in real time and as a summary average.

The Airspeed Calibration function is used to add position error corrections for a variety of pressure ports on the airplane to a reference pressure (such as trailing cone) and determine the speed of airflow over the pressure ports.

3.5 Utility

The ADAMS Utility functions are primarily used to aid the operator in getting the system ready for a particular in-flight test or condition. Some of these functions are also used to check the operation of the data acquisition system or prepare this system for flight. These functions generate special displays of PCM parameters and/or file records.

The Editor function is the primary function by which the operator may display, modify, insert or remove Data Base file records. This function may be used in flight; however, any time it is used, it is expected to be used to prepare for a specific test or condition.

The Help function generates special displays of instructive text which may aid the infrequent or novice operator to make commands controlling the various functions.

The Preflight function assists the operator to perform operational checks of PCM parameters prior to flight and to record the status of parameters checked. This function also performs noise checks on selected PCM parameters and/or compare groups of PCM parameters which may be expected to have the same value to point out deviations from the norm.

The Setup function is used to load the contents of PCM Decommutator and Word ID memories and/or verify the contents of these memories.

The Functional Test function is used to maintain records of checks made on PCM parameters by instrumentation engineers prior to the first flight of an airplane. This function also provides displays of text to instruct the operator on how to make these checks.

The Strain Gage Bridge Response function provides linear regression coefficients to correlate bridge outputs with applied loads during airplane structural calibrations.

The Loads Inertial Correction function computes loads inertia correction constants for each linear multiple equation performed by the loads function.

The Directory Dump function provides the operator with brief listings of Data Base record identifiers.

The Calibration Fit function will compute regression coefficients (linear single section, linear multiple section, or polynomial) for a set of data points obtained during a measurement calibration performed onboard a test airplane.

The Calibration Conversion function will convert lab calibration regression coefficients into coefficients usable by ADAMS in the absence of support from FTCS. This function will also combine calibration coefficients from two or more transducer components to produce a single set of calibration coefficients usable by ADAMS.

3.6 Support Functions

The ADAMS software includes certain "stand-alone" programs which may be run on the system in lieu of the normal ADAMS program. These programs are used to troubleshoot a malfunctioning system or prepare a new system for use.

A version of RMX/RDOS single user BASIC with several assembly language subroutines is supplied to allow the ADAMS user to implement small utility functions of his own design with a minimum of implementation flow time. The implementation of BASIC as a support function detracts from its usefulness because the ADAMS Monitor functions cannot be run at the same time. It is hoped that this can be corrected in future improvements to ADAMS.

Diagnostic programs are provided with ADAMS as necessary to troubleshoot and repair the ADAMS hardware. This includes the ROLM IDMS and those diagnostic programs provided by ROLM which are applicable to ADAMS.

Additional diagnostic programs for ADAMS unique hardware are added to the diagnostic diskette by Boeing. Boeing also plans to enhance the ROLM System Reliability Test by the addition of tests for ADAMS unique hardware.

A Memory Dump module similar to the RMX/RDOS core dump module can be added to the ADAMS software for use in development work. This allows core image files to be written to floppy disk during debugging and software testing. This module is generally not included in delivered systems.

4.0 DATA BASE

The ADAMS Data Base contains parametric data required to support the ADAMS software in the processing of flight test data. These parameters consist of data items subject to change between airplane models or between tests. The primary objective in designing ADAMS to include a Data Base was to make the software airplane and test independent.

The Data Base consists of several contiguous disk files. These are RDOS files which have been moved from floppy disk to fixed head disk prior to a test. Each file contains a logically related set of parameters. The type of parameter and the logical key are the determinants of which file will contain a parameter. The array of Data Base files is shown in Figure 9.

The MIT (Measurement Information Tables) Data Base file contains information necessary to obtain PCM data, convert the data to engineering units and display the data with standard Monitor functions.

The CONFIG (Configuration Information) Data Base file contains additional measurement information useful in preparing the Data Acquisition System for a test or troubleshooting a malfunctioning measurement.

The LIST Data Base file contains lists of up to 20 measurements which may be displayed by the Monitor functions.

The PRG (Program Information) Data Base file contains initialization and control information necessary to bring the Application functions into execution.

The TCP (Test/Condition Parameters Table) Data Base file contains parameters which define the particular test or condition (a portion of a test) to be performed on the airplane.

The APT (Airplane Parameters Table) Data Base file contains parameters which define the airplane under test.

The GC (General Calculations) Data Base file contains information which defines processing to be done by the General Calculations function. In general, each record in this file is an expression definition consisting of FORTRAN assignment statements.

The AV (Averages) Data Base file contains information which defines processing to be done by the Averages function.

The PCM Data Base file contains tables of control parameters to be loaded into the PCM Decommutator by the Setup function.

The WID (Word Identifier) Data Base file contains tables of control parameters to be loaded into the Word Identifier by the Setup function.

The KEYF (Key File) Data Base file contains ASCII strings to be treated as commands in lieu of actual keyboard entries.

The POS (Position) Data Base file contains list position information used by the ADAMS operator to sequence through the display of several lists of measurements.

The HELP Data Base file contains operating instructions and system information to be displayed by the Help function.

The EDIT Data Base file contains record format definitions for the other Data Base files. This information is used by the Editor function to display Data Base file information.

The LOADS Data Base file contains information which defines processing to be done by the Loads function. Each record in this file defines a "linear multiple equation" by which strain gage inputs are combined by the Loads function.

The RELOC (Program Relocation) Data Base file contains information used by the Job Controller to load and relocate Application functions.

The MSG (Message) Data Base file contains error message strings which are displayed on the operator screen in the event of an error.

The FT (Functional Test) Data Base file contains records of functional checks made on the Data Acquisition System prior to the first flight of an airplane. These records are maintained by the Functional Test function.

The FTST (Functional Test Support Text) Data Base file contains text consisting of instructions for completing functional checks and calibrations onboard the airplane prior to first flight.

The MISC (Miscellaneous) Data Base file contains information used by several Application functions such as display formats and initialization constants.

Each Data Base file has three major parts as shown in Figure 10. The first part of a Data Base file is the Preamble. The Preamble contains file identification and applicability information such as file name, airplane model, airplane identification, test identification, date and time of Data Base file generation or modification. The second part of a Data Base file is the Directory. The Directory is used to locate information within the Data Base file. The third and final part of a Data Base file is the Data. Each Entry in the Directory is associated with a single Data record. Both the Directory and Data areas are padded with sufficient space to allow addition of a reasonable number of new Entries and Records.

Each Directory Entry as shown in Figure 11 consists of a Record Identifier, a Record Pointer, and a Record Size. Record Identifiers are floating point numbers for some Data Base files and four ASCII characters for others. Record Pointers are double precision byte addresses of the beginning of the associated records

relative to the beginning of the file. The Record Size word contains the number of words in the associated Data Record.

Within each Data Record as shown in Figure 12, there may be several Data Elements. These may be floating point numbers, integers or ASCII strings. In some Data Base files, all Data Records have the same format; in other Data Base files the format is dependent on the Application for which they are used.

5.0 SUPPORT SYSTEMS

The ground based computer systems supporting ADAMS are unique to Boeing Flight Test and deserve some discussion. These systems are shown in Figure 13. The Airborne Data Systems Development Laboratory (ADSDL) is the software development facility for ADAMS. The Data Base Management System (DBMS), one of two systems referred to collectively as the Flight Test Computing System (FTCS), is used to maintain data base parameters to support all of the Flight Test data processes. The Instrumentation Sub-System (ISS) is used to generate transportable media (floppy disk) to transfer Data Base Files and Program files from the ground based systems to ADAMS.

5.1 Airborne Data Systems Development Laboratory

All the ADAMS software is developed in the Airborne Data Systems Development Laboratory (ADSDL). Permanent installations include two Data General Eclipse computers and two ROLM 1666 computers. The Data General Eclipse S200 and S230 share a 200-megabyte disk. Each computer operates in a dual program mode allowing four programmers to have full system capability. A four-drive AED 6200 floppy disk is connected to each computer and a system driver has been written and installed in the ARDOS and ZRDOS systems. ROLM software, including the ROLM FORTRAN Compiler, the Macro Assembler, the Relocatable Loader, and the RMX/RTOS System Generation program, have been purchased in source form, assembled, and installed on the system. This allows programmers to develop software to the core image form (save and overlay files). Programs are then moved to the floppy disk and moved to one of the two ROLM 1666 computers for debugging and software testing. Each of these computers is installed in a nearly complete ADAMS hardware complex (some peripheral equipment is shared). The ability to play back copies of flight data tapes allows a simulation of in-flight conditions. When programs have been satisfactorily demonstrated in the ADSDL and are ready for airborne use, they are transmitted to the Flight Test Computing System (FTCS). This is accomplished by the use of an RJE-HASP link and the running of the Data General HASP Emulator (HAMLET) on the Eclipse S230.

5.2 Flight Test Computing System

The Flight Test Computing System (FTCS) is a large and complex system based on an IBM 3033 computer. A complete discussion of this system is beyond this text; however, the important functions relating to ADAMS are worth noting. The most important function of FTCS relative to ADAMS is the maintenance of data base parameters. The Data Base Management System (DBMS) in FTCS controls a very large collection of parameters to support not only ADAMS but many other functions in Flight Test as well. The DBMS receives data interactively from several operators throughout Flight Test using menu driven display screens. On command from an operations engineer responsible for a

particular test airplane, appropriate data base parameters are transmitted to the Instrumentation Sub-Systems (ISS) to be made into Data Base files for use by ADAMS. Transmission is via RJE-HASP links.

In addition to the DBMS function, FTCS performs a sort of packet switching function to distribute ADAMS program files. These program files are treated as binary data sets and are held by FTCS only until valid reception has been acknowledged by each ISS. No transformation of the data is done by FTCS.

5.3 Instrumentation Sub-System

The Instrumentation Sub-Systems (ISS) are DEC PDP-11 computers which are used as intelligent output ports for FTCS. Other local functions are also performed in support of the Flight Test Instrumentation group. Each ISS includes a two-drive AED 6200 floppy disk. System drivers are not used for the floppy disk but rather local drivers in application programs do all I/O to this device. This allows formatting and initialization of floppy disks in RMX/RDOS format and the creation and writing of Data Base and Program files on these diskettes.

6.0 OPERATIONAL SCENARIO

The preparation and operation of the Flight Test data processing systems require coordination of many people, both within and outside of the Flight Test Engineering organization. Much consideration has been given in the design of all of these systems to the diversity of people who must interface with them.

6.1 Preflight

Requests for data are received by the Flight Test Engineering organization from other organizations throughout the Boeing Commercial Airplane Company. These typically include design, research and development, flight simulation and customer support organizations. The Flight Test Analysis engineers analyze these requests as well as current Federal Aviation Administration requirements for airplane certification to determine what parameters must be acquired and recorded and what data processing must be done to satisfy the total data need. As a result of this analysis, a list of instrumentation requirements is developed using the DBMS. Initial receipt of such a list on the ISS prompts Flight Test Instrumentation engineers to begin designing and implementing installations of Data Acquisition and ADAMS hardware. The Analysis engineer may also request new data processing capability for either ADAMS or TDPS. The Airborne Data Processing group analyzes airborne data processing requests and designs and implements new functions as necessary. After satisfactory demonstration of software in the lab, released software is transmitted from ADSDL to FTCS and forwarded to ISS. The Analysis and Instrumentation engineers must work interactively with the DBMS to build data base parameters necessary to support Data Acquisition, ADAMS, DPGS and TDPS. Meanwhile, Flight Test Operations engineers prepare a plan of test which is the script for directing the test.

ADAMS is made operational as soon as the installation on an airplane is complete. Diagnostics are run to prove operability and then the system software and Data Base files are moved from floppy disk to fixed head disk. ADAMS is used to check the operation of the Data Acquisition System. Checks

are made on each PCM parameter using the ADAMS Monitor functions and records are kept with the aid of the Functional Test function. On-board calibrations are made if necessary, aided by the Calibration Fit and Strain Gage Bridge Response functions. Throughout the installation phase, data base parameters in FTCS are continuously updated by interactive input.

Shortly before first flight and each flight in the test period, the most current Data Base files are installed on the ADAMS disk. Within the data base are indicators of which PCM parameters are required for the current test. Operational checks are made on each of these parameters. The Preflight function aids in this process and is used to maintain a record of the process. When each parameter has been checked and all airplane systems are ready for flight, the airplane is released for flight.

6.2 Flight

Upon release of the airplane, the flight crew boards the airplane. In addition to the pilot, copilot, and flight engineer, the flight crew typically includes Operations Instrumentation and Analysis engineers as necessary to direct, monitor, and evaluate the test. In many cases, representatives of requesting organizations or the FAA are also included.

As the pilot, copilot and flight engineer check the airplane for flight, the Instrumentation and Analysis engineers check the Data Acquisition System and ADAMS. Data Base parameters are displayed and verified. Application functions used for preflight are replaced by Monitor and Data Analysis functions appropriate for the current flight.

When all systems are ready for flight, the flight data recorder is turned on, the engines are started and the test begins. Since all of the systems required to accomplish flight testing are self-contained on the airplane, testing is not restricted to the local area. If the weather or field characteristics in the local area are not suitable for the current test, the airplane may be flown anywhere within its normal flight range.

The conditions for each test (airspeed, altitude, engine settings, etc.) can be accurately adjusted and verified using the Monitor and Data Analysis functions. If the conditions cannot be met, time is not wasted performing tests which might produce questionable data. This type of cost-saving decision can only be made because accurate processed data is rapidly available whenever and wherever the airplane flies.

The conduct of the test is closely monitored to insure that all testing is done strictly according to plan. The real-time data reduction performed by ADAMS condenses a large number of measured parameters to a few of the most significant indicators of test conduct and airplane performance. This enables a few engineers to monitor hundreds of parameters in real time.

As each item on the test plan is completed, the results of testing may be quickly compared to results of previous tests and to design predictions. Output data formats have been designed to maximize the similarity between ADAMS output and ground based final data output. This eases the comparison task. In some cases, target data values are even contained in the ADAMS Data Base to enable Application functions to make these comparisons. If test results should

fail to meet design predictions, decisions can be made in flight on whether to continue the line of testing or to suspend the testing until corrections can be made. Again, cost savings are realized because early decisions are made which can only be based on processed data.

6.3 Post-Flight

After completion of a test flight, the flight data tape is normally taken to the Data Processing Ground Station to begin the stripping of usable data from the tape. ADAMS may be used either on the airplane or in a lab to play back the flight data tape. This allows additional analysis of the flight data to determine the extent of usability. Requests for processing by DPGS and the Test Data Processing System are thus minimized, resulting in additional cost savings.

Selected data which has been extracted from the flight data tape and transmitted to TDPS is processed by applications similar in function to ADAMS, but much larger in scope. The absence of time constraints and the availability of very large storage allow much more complex data transformations to be accomplished.

With final data in hand, the Analysis engineers prepare reports to the organizations which requested the data, including the FAA. Meanwhile, on all but a few Boeing owned airplanes, the special equipment is removed and the airplane is prepared for delivery to a customer airline.

7.0 FUTURE SYSTEMS

The future of airborne data processing in Boeing Flight Test is a steadily increasing demand for processing capacity, speed, and reliability. We believe that this demand can be best met by the introduction of a multiple processor system. A possible configuration is shown in Figure 14. Other concepts are being evaluated. Microprocessor technology is being viewed with considerable interest because this could allow the configuration of a system particularly tailored to a specific Flight Test application. A combination of microprocessor elements with the medium scale ROLM 1666 is likely.

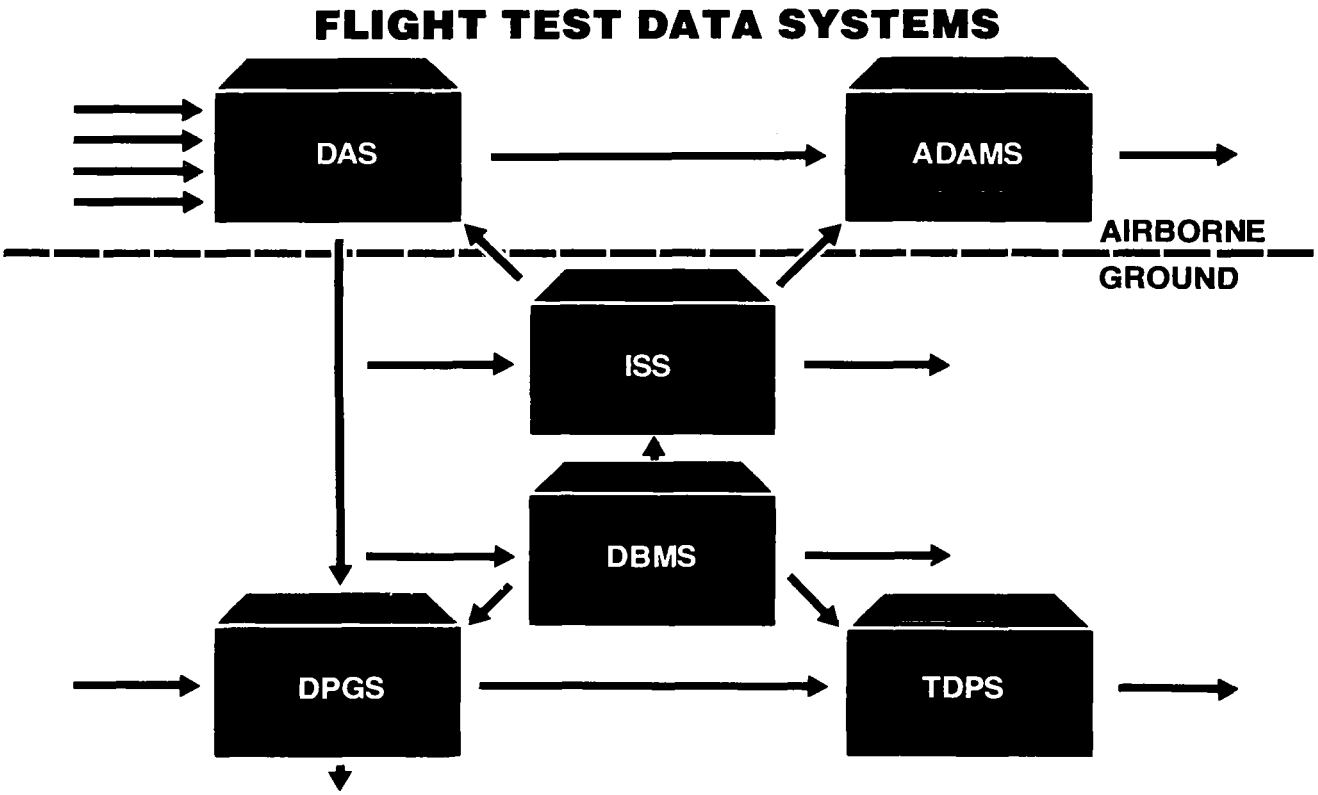


Figure 1

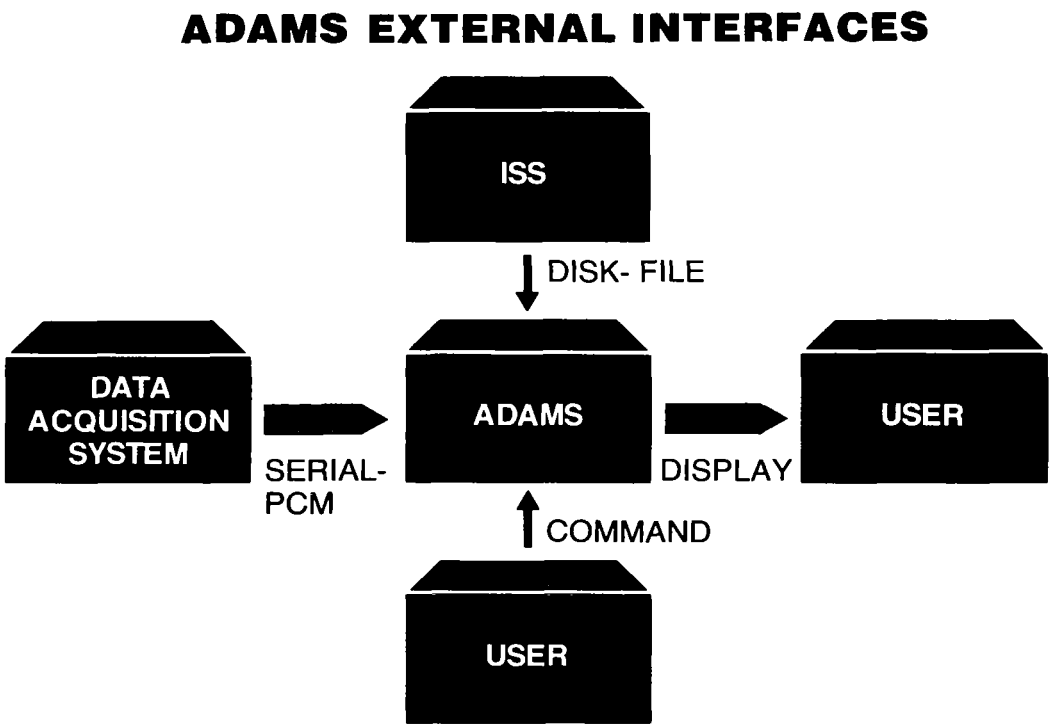


Figure 2

ADAMS HARDWARE

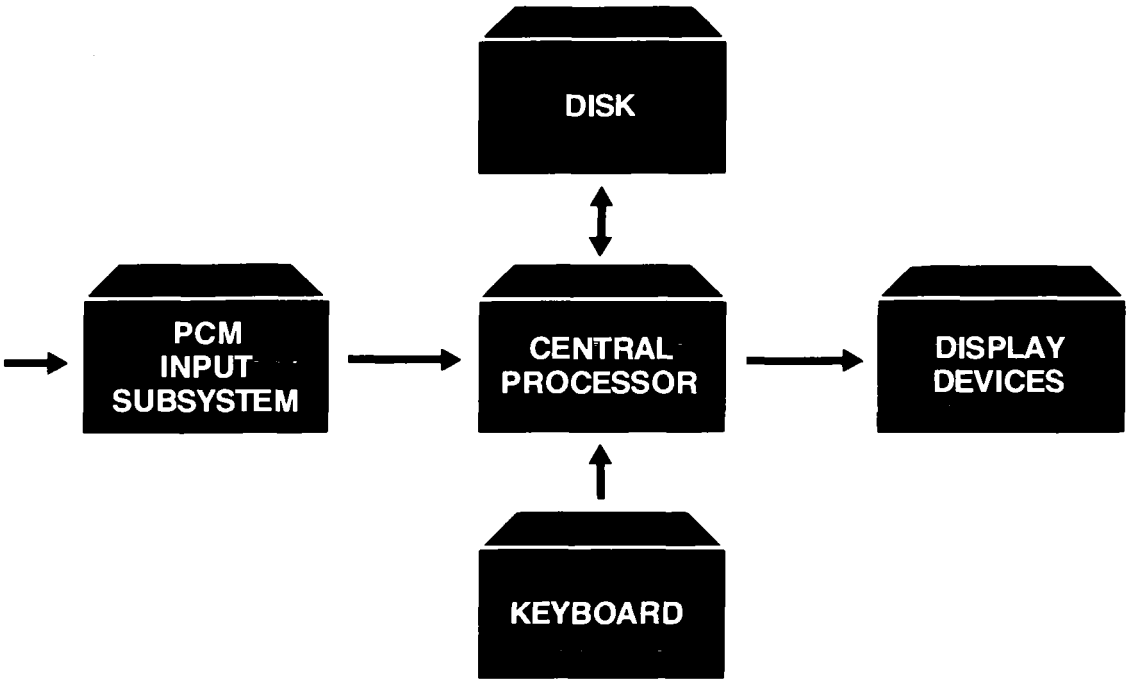


Figure 3

ADAMS PCM INPUT SUBSYSTEM

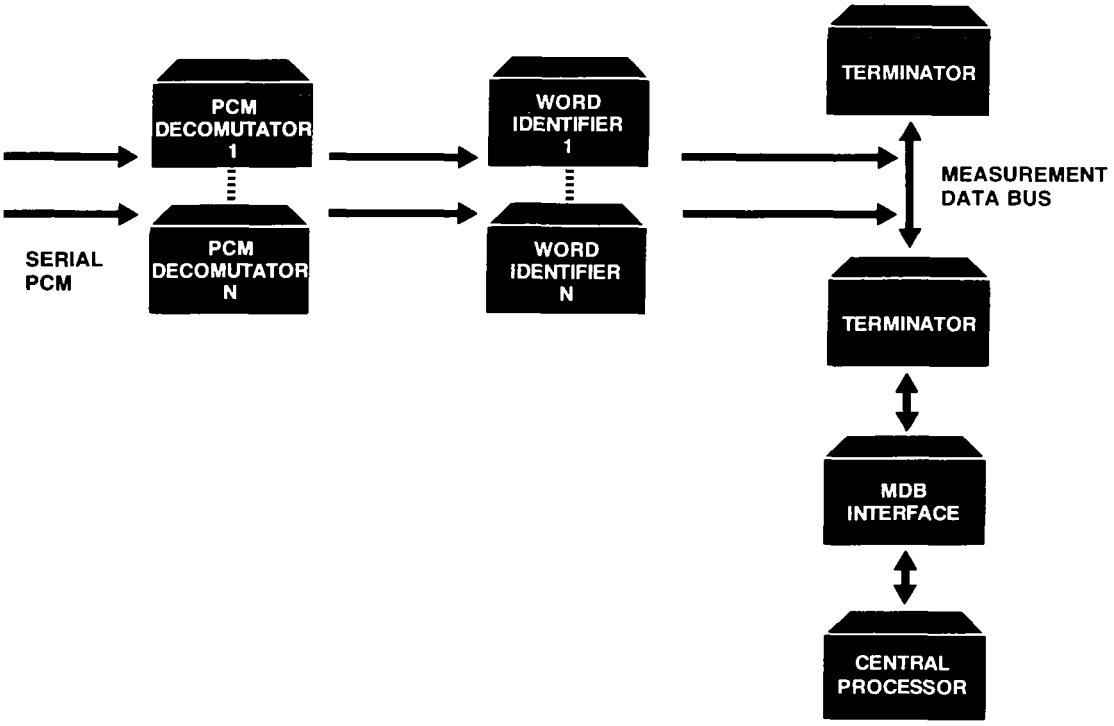


Figure 4

ADAMS DISPLAY DEVICES

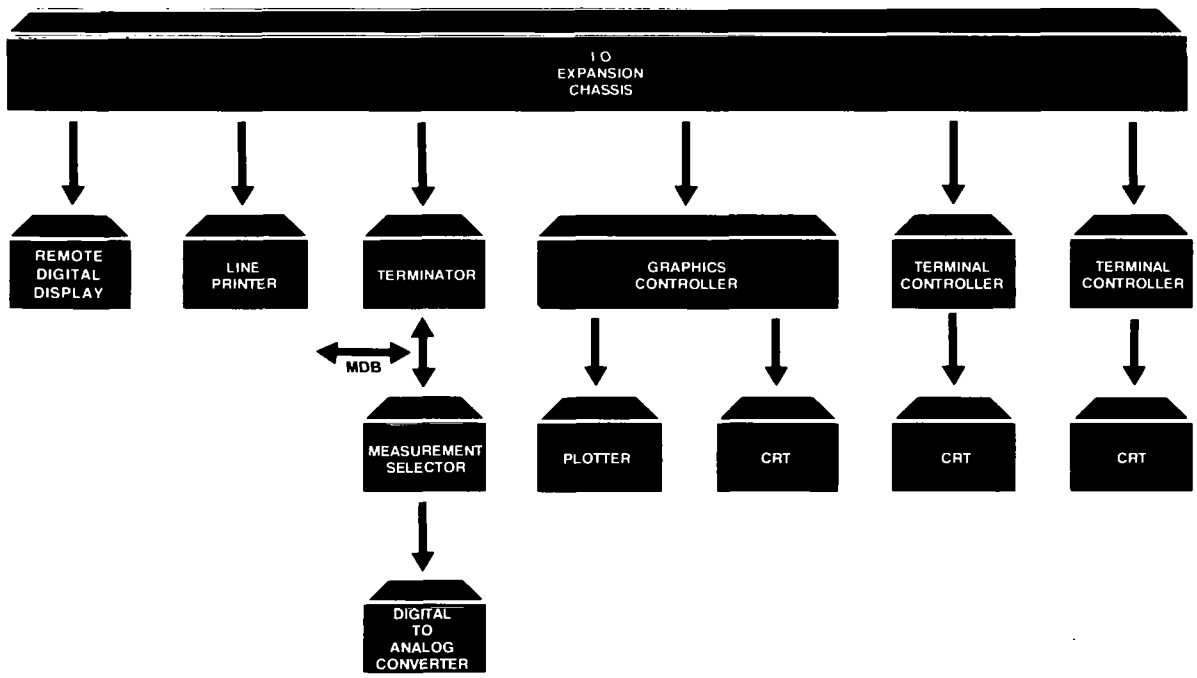


Figure 5

ADAMS SOFTWARE

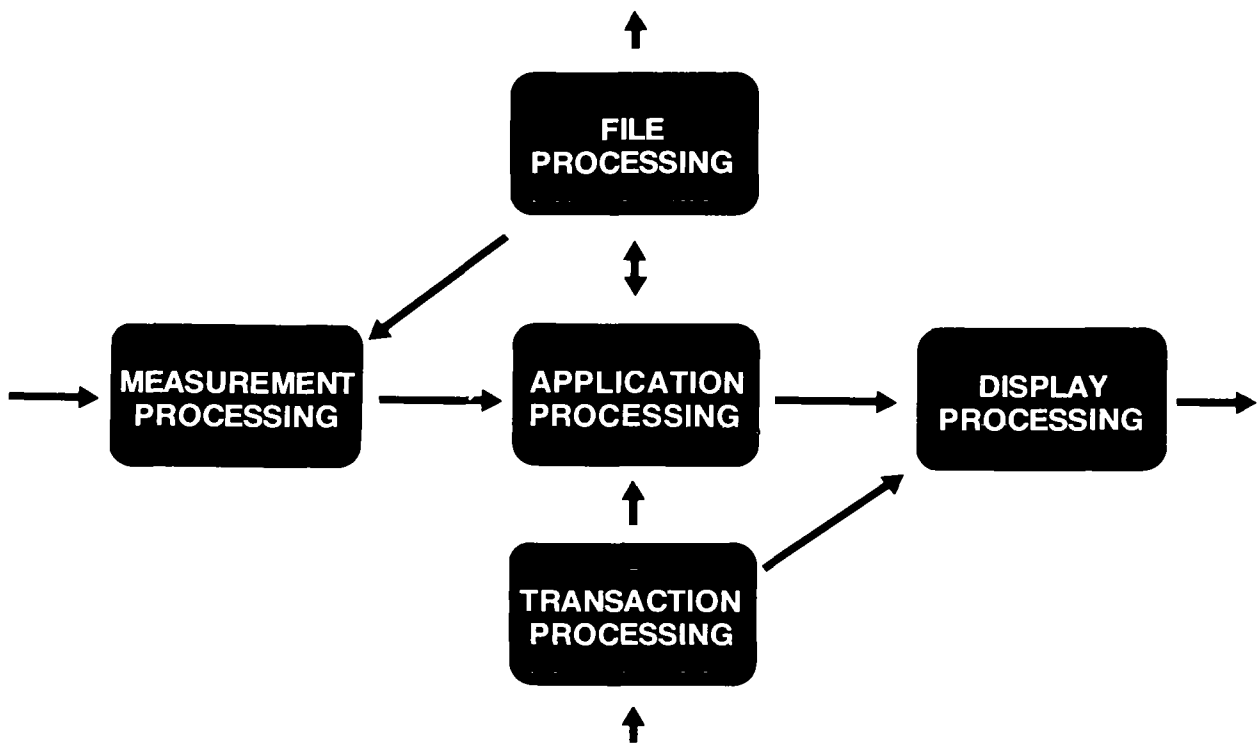


Figure 6

ADAMS APPLICATIONS

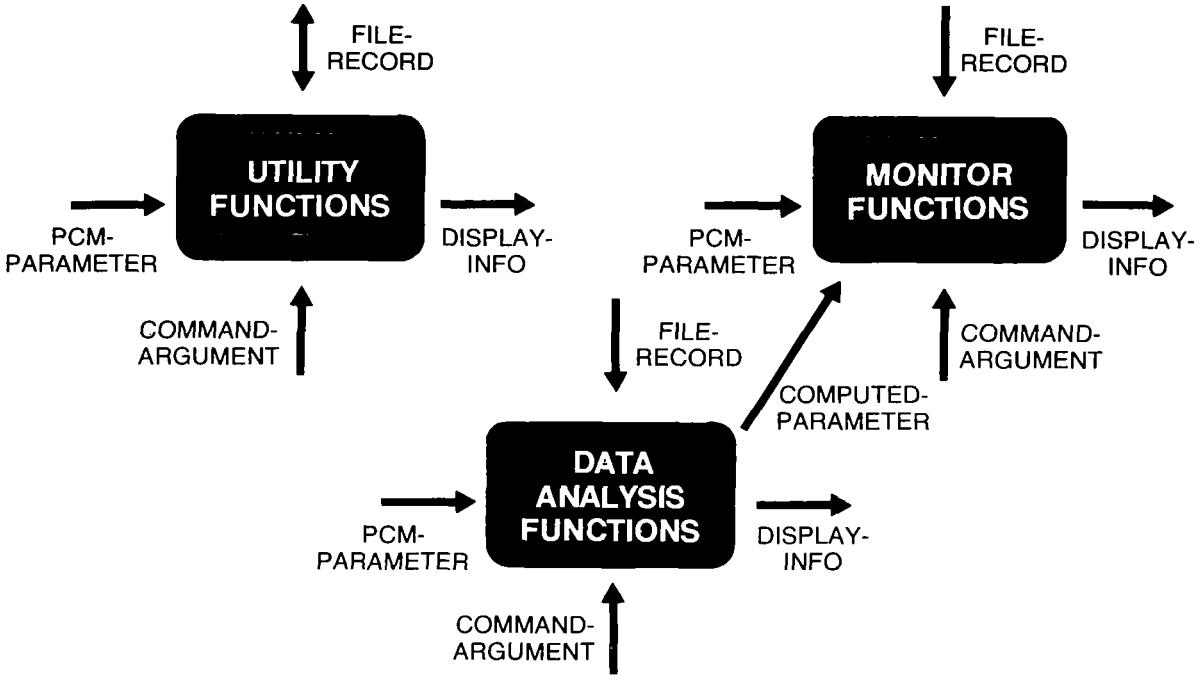


Figure 7

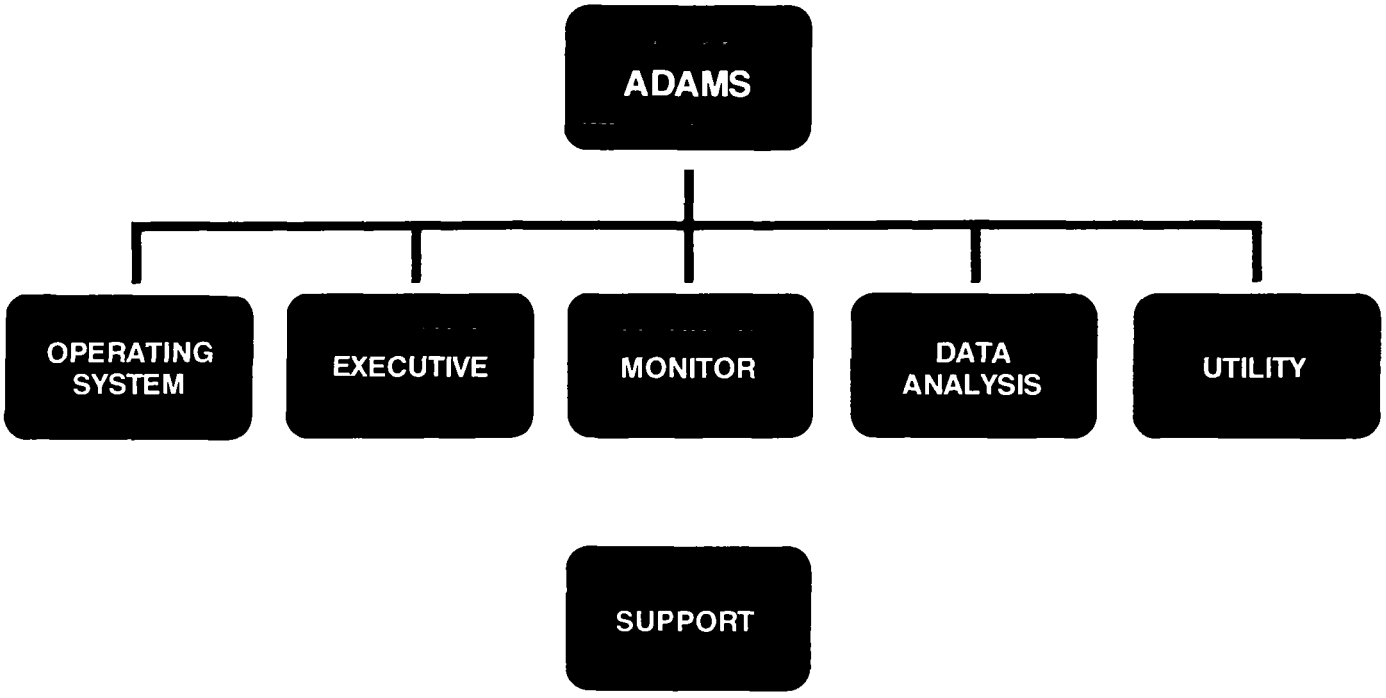


Figure 8

ADAMS II DATA BASE

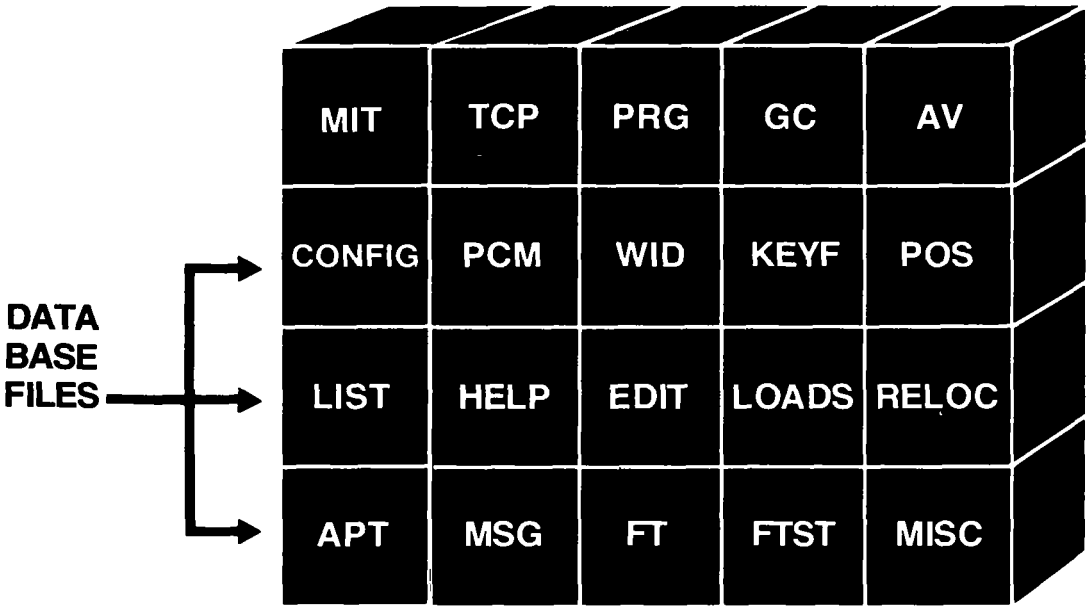


Figure 9

ADAMS DATA BASE FILE

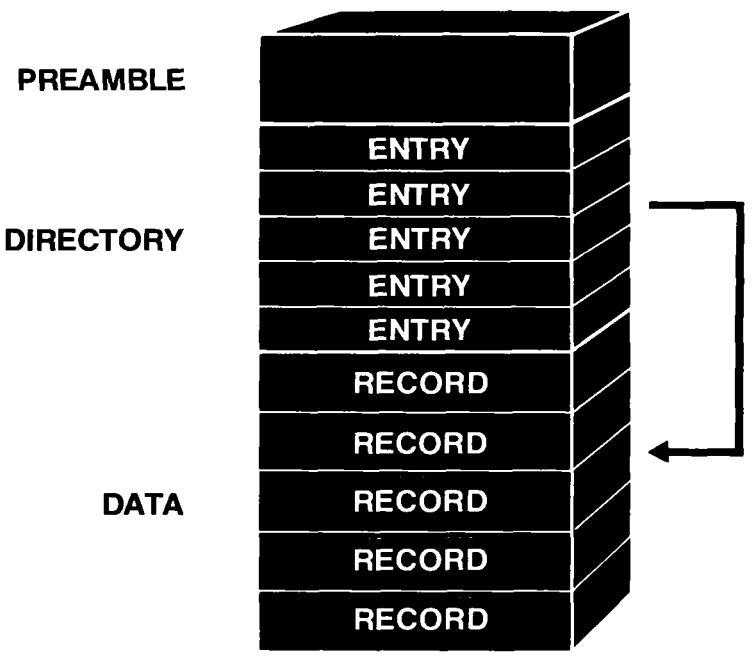


Figure 10

ADAMS DIRECTORY ENTRY

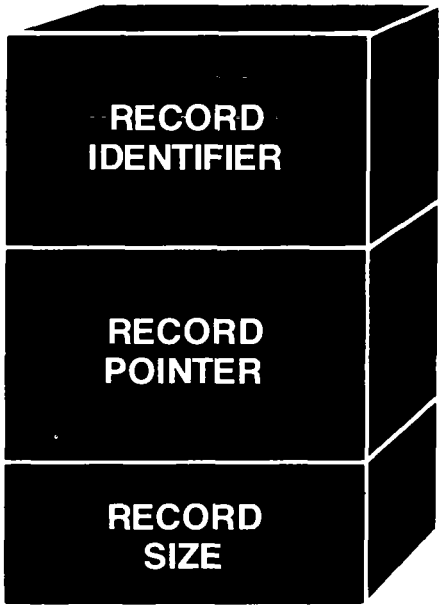


Figure 11

ADAMS DATA RECORD

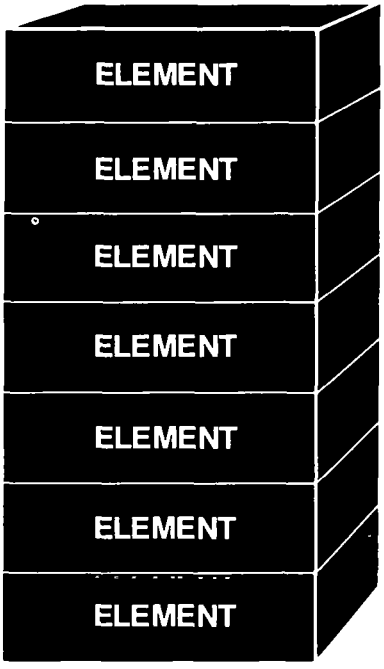


Figure 12

ADAMS SUPPORT

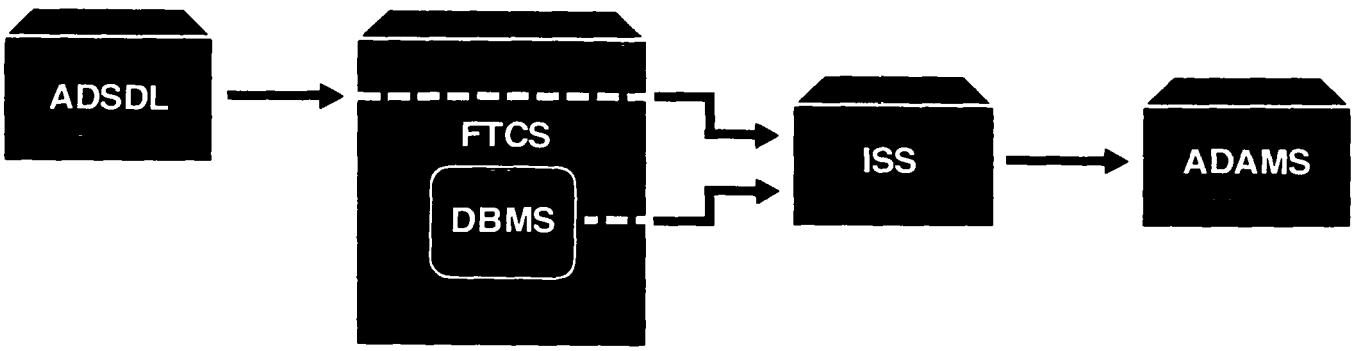


Figure 13

ADAMS MULTI-PROCESSOR SYSTEM

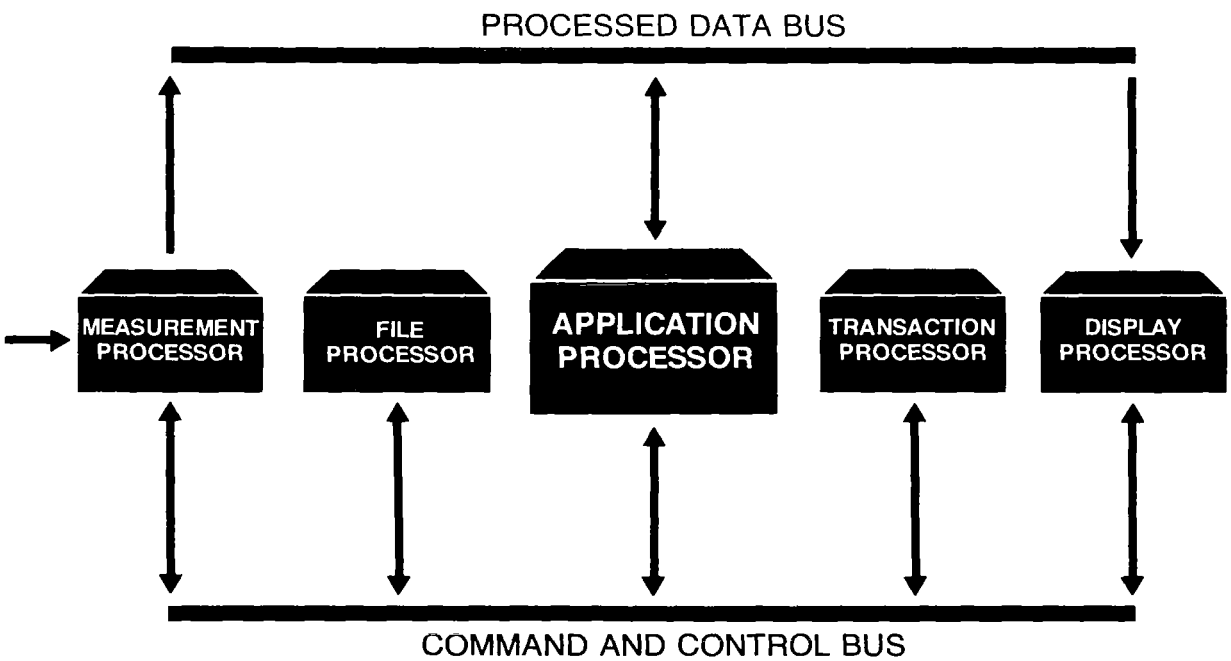


Figure 14

ADAMS EXECUTIVE & OPERATING SYSTEM

W. D. Pittman
Boeing Commercial Airplane Company
Seattle, Washington

ABSTRACT

The ADAMS Executive and Operating System is a multi-tasking environment under which a variety of data-reduction, display and utility programs are executed. This environment provides a high level of isolation between programs which allows them to be developed and modified independently.

INTRODUCTION

The Airborne Data Analysis/Monitor System (ADAMS) was developed to provide a real-time data monitoring and analysis capability on board Boeing commercial airplanes during flight testing. It inputs sensor data from an onboard data acquisition system and converts it to engineering units data, derives airplane performance data by applying transforms to the collected sensor data, and presents this data to test personnel via various display media.

ADAMS is a real time transaction-oriented computing system. ADAMS operators input processing requests at the system consoles as necessary to evaluate flight test conditions (a simplified model of ADAMS is shown in Figure 1). Operator requests are input to the system executive which then schedules the requested processes. Active processes retrieve operator commands, airplane sensor data and support data via the executive, send processed data to the executive for subsequent input to other active processes, and send formatted data to display devices via executive device management software. These processes perform functions such as the collection and display of engineering units data on various output media (line printer, CRT display, graphics display, strip chart recorder); collection and reduction of real time data (data averaging and integration, computation of derived parameters); and support processing (support data display and modification, hardware setup, system checkout).

DESIGN CONCEPT

ADAMS is based upon the concept of functionally independent parallel processes which are initiated and controlled by explicit operator commands. Although dozens of processes are available to be invoked by the operator, typically only a subset of the total is selected to run concurrently. The demand for system resources made by these processes is so varied that a suitably versatile program environment was deemed necessary. Since it was expected new functions would be added to the system on a regular basis and that existing functions would be updated and modified frequently, it was also a requirement that the functional capability of ADAMS be quickly adaptable and easily extendable. In order to meet these requirements, it was decided to develop the system around a vendor-supplied operating system and implement ADAMS functions as independent programs written in a high level language. Since the ROLM 1666 was selected as our system processor, RMX/RDOS and ROLM FORTRAN became the operating system and high level language upon which the system was based, although it was expected that the operating system would have to be enhanced or modified in order to implement the program environment which was desired.

One of the goals during the development of ADAMS was that the various application programs be very loosely coupled with one another. Another was to isolate the applications from the hardware and operating system and provide them with very high level interfaces with which to communicate with their environment. The programs would use these interfaces to fetch and store engineering units data, communicate with the operator, fetch and store support data, and output to display devices.

SYSTEM DESCRIPTION

The management of ADAMS application programs is accomplished by the transaction processor, scheduler and loader, as shown in Figure 2. The transaction processor accepts and validates user commands for requested processing. If the command was a legal request and the requested process was not active, then the transaction processor will read the process descriptor from disk and install it into the active process table. The process descriptor contains information such as iteration rate, priority, overlay name, status information, and program location, and is used by the scheduler and loader to execute the process as required. The scheduler (via real time clock and measurement data interrupts) determines when each active process is ready to begin execution. If the overlay defined for the process is not in core when the process is ready, then the loader will load the overlay into memory and the process will begin execution, otherwise the process will begin execution when ready.

Very early in the design of ADAMS it was recognized that main memory would be a very limited resource. It was anticipated that all of the required ADAMS application programs would not fit in available memory (64K). Furthermore, the likelihood that those programs required to run in parallel would not fit in memory together was very high. The most obvious solution to the problem was to buy the required additional memory; however, it was felt that buying the memory required for 32 systems would not be cost effective. Another alternative was to use the disk overlaying capability provided by the RMX/RDOS operating system, which was rejected because it was too limited to support our processing requirements.

The approach which was adopted relied on a run time program loader to swap ADAMS programs between disk and main memory (Figure 3) and a scheduler to execute the programs. When a program is ready to execute, the loader loads the program's common storage block and code into memory, swapping out any idle programs if necessary to make room. The loader adjusts all references by the program to the common storage block and external routines.

The program scheduler uses many of the facilities of RMX/RTOS to start and maintain the execution of ADAMS programs (Figure 4). An RMX/RTOS task is defined for each active program beginning at the scheduler starting address. When the task begins execution, it calls the loader, which installs the appropriate overlay into memory. The scheduler then executes the overlay as a subroutine, after which time it suspends until restarted by the real time clock handler when its cycle time has expired or by the Measurement Number Data Bus handler when a specified data item is input to the system. Appropriate status words are maintained for each program which allow the loader to move programs to and from disk as required for execution. (Of course if all executing programs fit into available memory, then swapping is not required.) A kill processor releases all system resources held by a program when it is terminated and is initiated either by an **UNLOAD** command received from the operator, or because the program finished executing a non-cyclic overlay and no cyclic overlay was specified, or because a software malfunction was detected in the program.

All data flow to and from ADAMS application programs is controlled by the system executive. Programs are not allowed to directly communicate with each other, nor are they allowed to communicate with the system or the external environment except through the interfaces previously mentioned. These interfaces, which are accessed via FORTRAN subroutine calls, define the data flow through the system and provide a consistent and simplified access method for system resources (Figure 5). Isolating the programs from one another allows them to be developed independently and reduces the risk of unwanted side-effects when application, executive, or operating system software is modified. Isolating the programs from the system resources reduces the risk of the inadvertent corruption of those resources. The access method to system resources reduces much of the effort required to develop application programs and defines a simplified conceptual model of the program environment.

The Measurement Data Generator is the conduit through which all cyclic data is passed through the system. It performs two functions for application programs: fetch engineering units data and save engineering units data. The fetch function retrieves a previously saved data value or fetches the sensor data (which is continuously being DMA'ed into memory) and converts it to engineering units. Conversion and calibration information for each sensor is stored in the support data base. The save function stores a data item for subsequent retrieval by other programs.

The Device Manager interfaces the application programs to the various ADAMS devices such as the line printer and system consoles. It provides high level FORTRAN calls to communicate with the devices and handles contention among application programs for those devices.

The File Manager provides a high level interface to the system disk for application programs and other executive functions. It is an enhancement of the limited RTOS disk support facility and provides routines to create, delete, open, close, read and write disk files. All disk data transfers are buffered by the system in a manner similar to that done by RMX/RDOS.

The Data Base Manager implements the access method by which programs fetch, store and modify the system support data. Each item in the support data base is identified by a unique key which is used by the program to request action on the item.

The application programs alter their own or other programs' execution status via calls to the Job Controller. These calls are used to start and send commands to other programs, change their own or other programs' iteration rate, unload (kill) themselves or other programs, chain to new overlays, or adjust the size of their common storage.

ADAMS is based upon a modified version of the RMX/RTOS operating system. Most of the features of that operating system have been retained, but many of the device drivers have been modified, a disk file management facility was added, the system error and trap handling facility was expanded, and task calls have been added or modified.

Many of the existing RTOS device drivers (TTY, LPT, ALM) have been or are being modified to improve their efficiency or adapt them to accommodate special ADAMS devices. A driver to handle a Boeing-designed data bus interface was implemented and integrated with RTOS, and a real time clock handler was installed to facilitate the scheduling of ADAMS programs.

The system error and trap handler was added to allow ADAMS to attempt recovery from software malfunctions. It gains control of the CPU when a processor trap or system

error is detected (stack overflow, unimplemented instructions, jump zero, etc.). If the error occurred while executing an application program, then that program is killed and all system resources held by that program released. If the error occurred while executing the executive or operating system, then the system is reinitialized (rebooted) and the operator informed of the malfunction.

Two task calls, LOK and UNLOK, have been added to allow processes and system resources to be locked and unlocked. The inclusion of these calls allows the locks to be cleared if the requesting task is killed. Other task calls have been modified to reduce system overhead.

In addition to the executive and operating system software described previously, ADAMS relies on three support programs (Figure 6). The first of these is a system generator which executes on a Data General Eclipse minicomputer. It uses a version of the ROLM relocatable loader and a Boeing developed utility program to link and load all system and application program object modules. It produces an executable save file, a system overlay file, and an application program overlay file. The system overlay file contains position independent system overlays, while the program overlay file contains program overlays and relocation information. The second of these programs, the bad block detector, executes on the ROLM 1666 processor. It searches the bad block pool on the fixed-head disk for bad blocks and sets the appropriate bits in MAP.DR to prevent those blocks from being used. This is necessary because the RMX/RTOS disk driver does not do bad block mapping. The third of these programs, the system initializer, runs under RMX/RDOS on the ROLM 1666. This program searches the disk for all files required during execution of ADAMS and inserts an entry for each into the RTOS disk table in the ADAMS save file, then "boots" the save file, bringing the ADAM System into execution.

SYSTEM DEVELOPMENT

During 1979, a prototype of ADAMS was developed which executed in the RMX/RDOS environment. This version was installed on a test airplane early in 1980 to make an initial evaluation of the system. Even though only a small subset of our application programs had been developed up to that point, it was very clear that the performance of the system was far below what was required.

We had fortunately anticipated the need to analyze system performance and had developed instrumentation hardware which allowed us to measure the performance of a program executing in the 1666. This instrumentation enabled us to sample the program counter in the 1666 and thus produce a histogram which displayed a distribution of time versus memory location.

Using the output of our instrumentation we were able to verify that the CPU was saturated and that about 80% of the time was being spent in RMX/RDOS system space, the majority of which appeared to be in swapping system overlays and handling serial I/O. Since by this time ROLM had released the RMX/RTOS operating system, the decision was made to abandon RMX/RDOS and convert the system to run under RMX/RTOS.

The similarities between RMX/RTOS and RMX/RDOS greatly aided the conversion effort, although numerous bugs in the released version of RMX/RTOS required a great deal of effort by our own software engineers to isolate. (We have since decided to maintain our own version of RMX/RTOS.) Although a disk file management capability had to be developed, the ADAMS Executive was converted and was executing under RTOS within three months. The isolation of the application programs from the operating

system had also been a great advantage; except for minor modifications required for one or two programs, the application programs were able to execute in the new environment.

The improvement in performance was dramatic: operating system overhead was reduced from 80% to about 50%, and memory usage of the Executive and Operating System was reduced from about 50K to about 32K. Additional optimization to both the operating system and executive since then has improved their performance to a reasonable level.

FUTURE DEVELOPMENT

Experience gained during the development and initial use of ADAMS has revealed limitations in system capability, while ever-increasing requirements for airborne data processing will require improved system performance. Three approaches are now being taken to improve system capability: enhancement of 1666 resident software, increasing the power/performance of the 1666, and distributing the processing load to additional computers.

The 1666 resident software will be enhanced to improve the reliability of the system. The memory protection features of the 1666 will be used to further isolate application programs from each other and from the system executive, thus reducing the risk of an aberrant program corrupting the system. Improvements to the system build process, consisting of a special relocatable loader to build program overlays and a utility to edit the program overlay file, will reduce the time required to generate systems and improve the maintainability of the system.

Development is already underway to improve the performance of the 1666 processor. Since a significant portion of ADAMS' CPU resources is spent servicing the various output displays, the decision was made to develop intelligent general purpose interfaces to handle the system I/O processing. These interfaces, which are planned to replace the TTY, ALM and printer interfaces, will be programmable to enable them to be adapted to a variety of devices, and will handle data transfers to and from the 1666 memory via DMA.

Recent requirements for airborne data processing have increased to the extent that our present 1666 CPU-based system is seriously underpowered. As an example, requirements for one new application program call for computations performed at a rate which is beyond the capabilities of the 1666. In order to be able to meet these and expected future requirements, we have decided to develop a multiprocessor architecture for ADAMS, as shown in Figure 7. This architecture will feature two high-speed buses: a cyclic data bus and a block data bus. The cyclic data bus, developed by Boeing, will carry cyclic sensor and computed data. The block data bus, the Ethernet bus developed by Xerox, will carry burst-type data (such as commands, support data, interprocessor communication, etc.). The ADAMS Executive will be extended to handle the additional processors, which will be connected across the two buses. The 1666 will remain in the system and will manage system resources, and perhaps display processing, while the data reduction and analysis programs will be distributed to other processors, for which we are currently planning to use the new generation of 16 and 32 bit microprocessors.

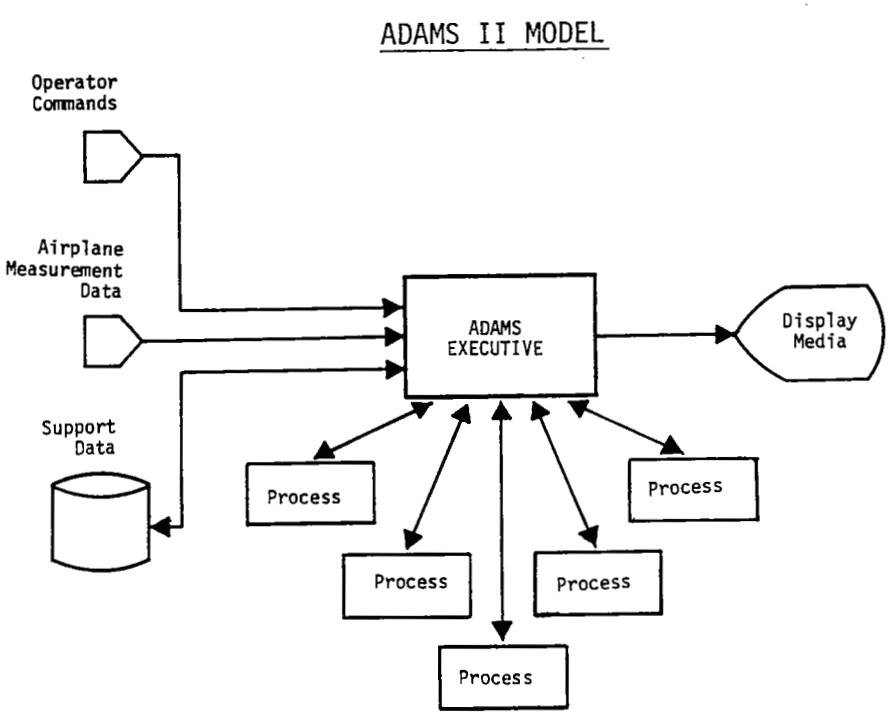


Figure 1

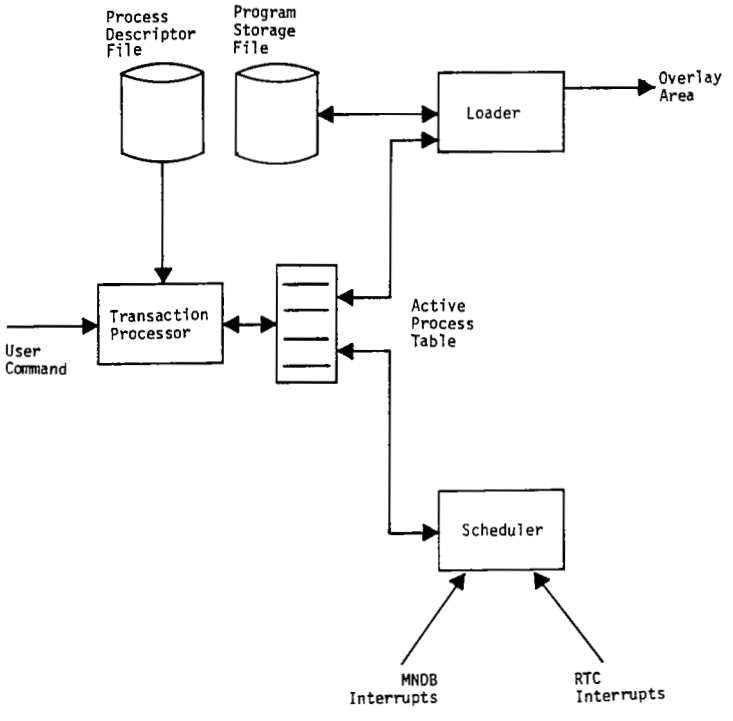


Figure 2

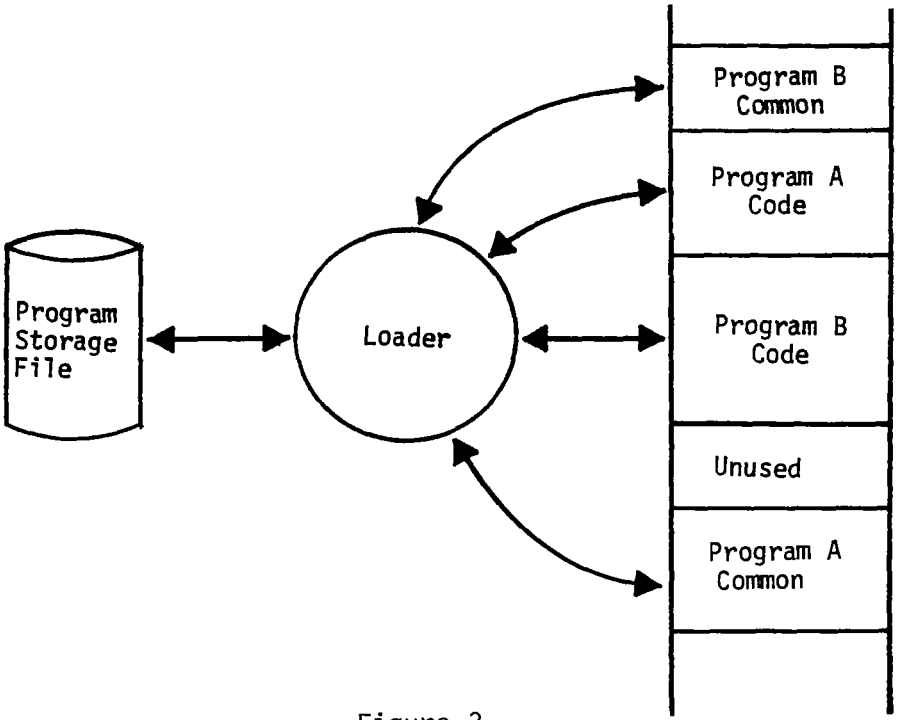


Figure 3

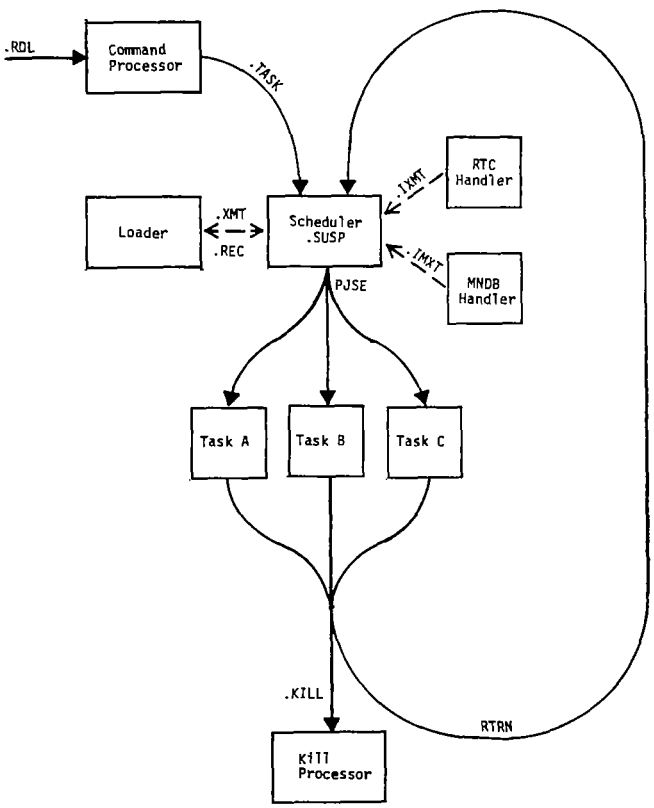


Figure 4

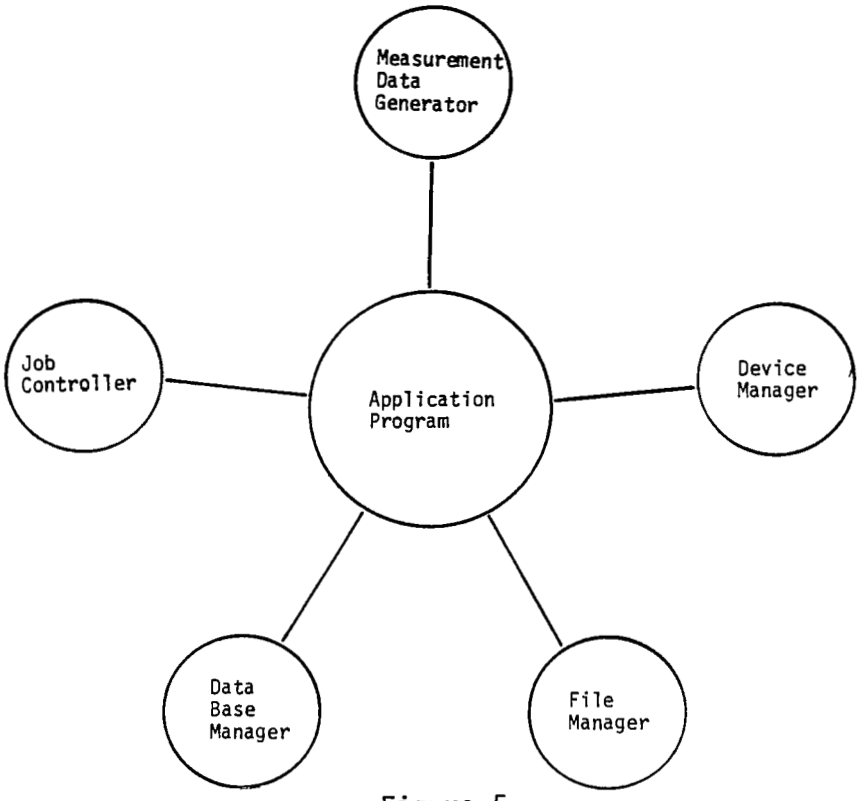


Figure 5

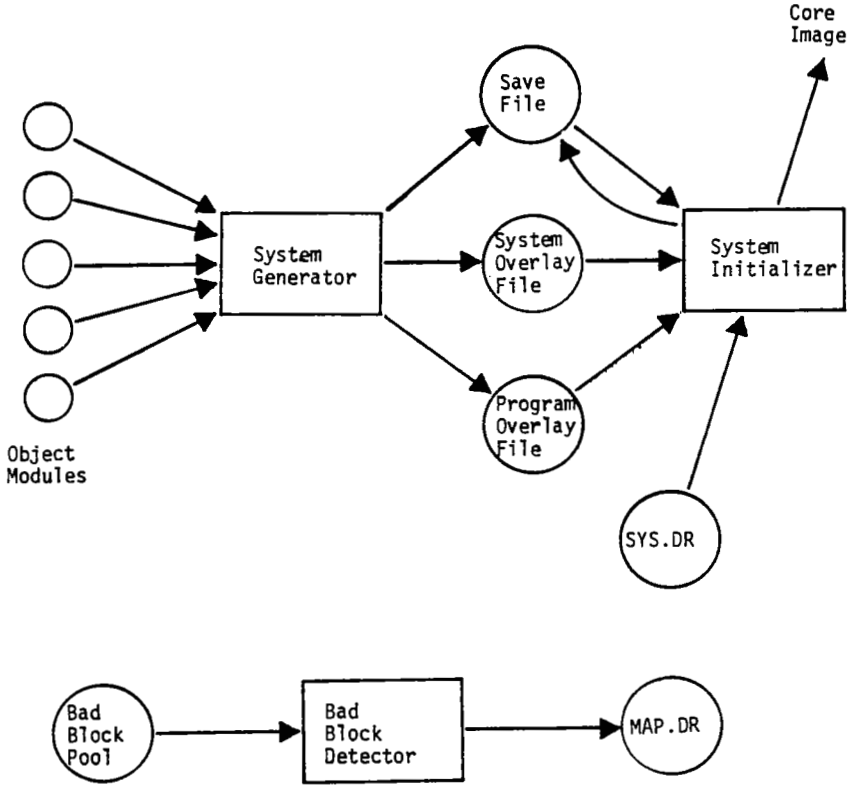


Figure 6

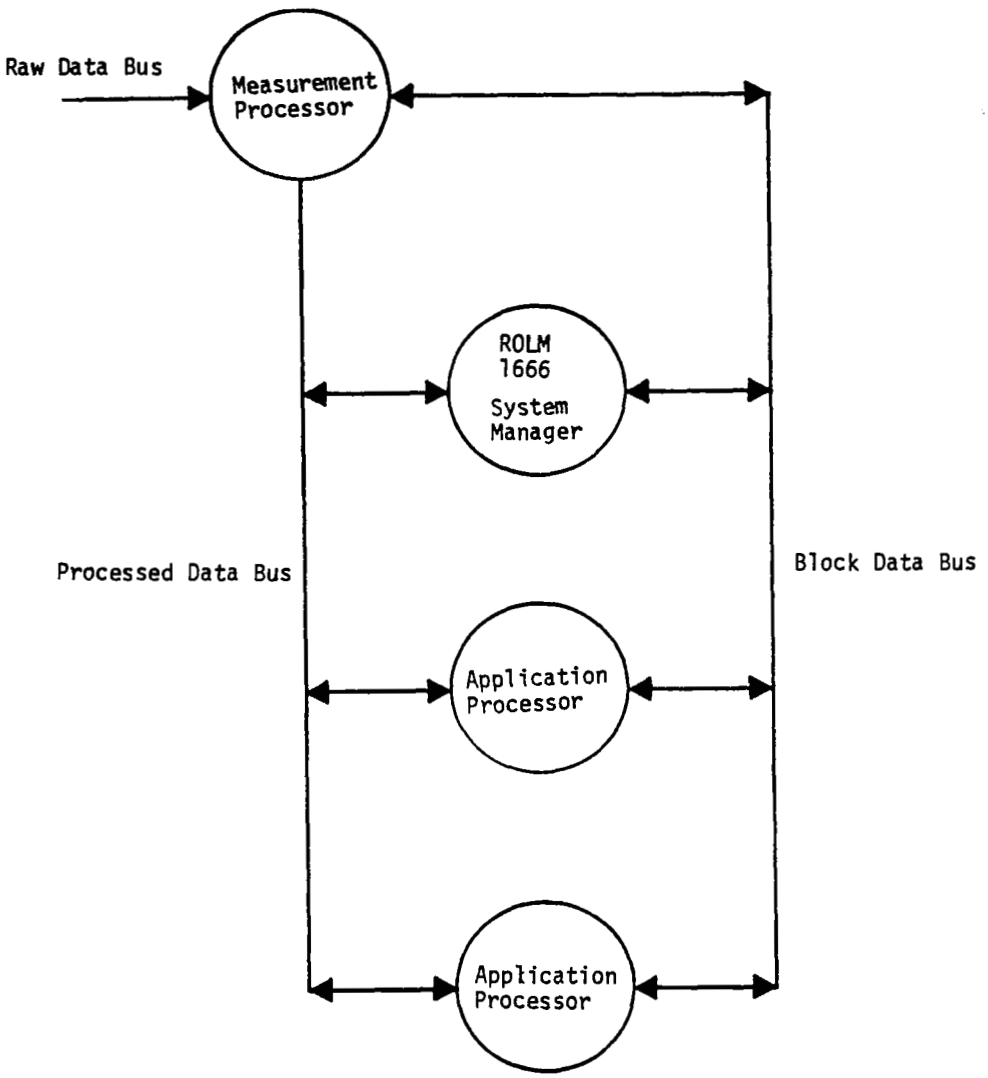

Raw Data Bus
Measurement Processor
ROLM 1666 System Manager
Processed Data Bus
Block Data Bus
Application Processor
Application Processor

Figure 7

SYSTEM PERFORMANCE ANALYZER

H. R. Helbig
Boeing Commercial Airplane Company
Seattle, Washington

The System Performance Analyzer (SPA) was designed to provide accurate real time information about the operation of complex systems. It is currently being developed for use on the Airborne Data Analysis/Monitor System (ADAMS), a ROLM 1666 based system used by Flight Test.

It uses an external processor to operate an intelligent, simulated control panel. Also provided are functions to trace operations, determine frequency of use of memory areas, and time or count user tasks in a multi-task environment. This augments the information available from the standard debugger and control panel, and reduces the time and effort needed by ROLM 1666 users in optimizing their system, as well as providing documentation of the effect of any changes.

This paper discusses the design of a System Performance Analyzer, a tool to evaluate the operation of a relatively complex computer system. It provides information on the operation and state of the system under study.

The System Performance Analyzer (SPA) is being developed for use with the ADAMS (Airborne Data Analysis/Monitor System) currently in use by the Boeing Company. It is an attempt to integrate the tools now used to provide information about the system. These tools are the control panel, the software debugger, a Boeing-developed timing and instruction counting histogram analyzer, and a commercial trace analyzer.

The ROLM 1666 Control Panel provides a number of functions useful in testing the ADAM System. The address display indicates the area of program running. Status lights show the mode of operation. The data display can be programmed to provide information. Hardware breakpoints can be used to stop program operation without loading the debugger. Unfortunately, it is somewhat awkward and time consuming to use.

The software debugger is also used extensively for system development. It allows the user to examine and change memory locations, accumulators and registers. It also provides eight breakpoints for interrupting a program at a given memory address. It has a number of limitations that severely limit its usefulness. It consumes system resources, does not provide time-dependent information, and often a program will run differently under breakpoint control than when running normally.

In an effort to provide more information about the dynamic operation of the system, Boeing developed a histogram and timing analyzer. This is a hardware box that connects between the control panel and the CPU of the system and allows a second computer to monitor the system operation (see Figure 1). This box provides two functions. In the histogram mode, the analyzer CPU periodically samples the instruction being fetched by the system CPU. A map of memory usage is then built up. This allows the user to see if excessive time is being used by a small section of programming. The timing mode allows the user to monitor the time necessary to run a section of code with a resolution of ten microseconds. This provides the user with the knowledge of what procedures need streamlining. Although it provides only limited information, it demonstrated the ease of use and real time information available by coordinating computer control.

A trace analyzer provides a history of operation. By monitoring the bus it becomes possible to see what caused the arrival at a given state. Although this is a useful tool, once again difficulty of use and interpretation limits it to special situations.

The usage of the previous tools pointed out certain things:

1) The correct tool must be used to solve a given problem (i.e. the software debugger is not useful for solving timing problems).

2) The tool must be easy to use (i.e. the trace analyzer, which must have a number of lines individually connected, is rarely used).

3) It must be easily interpreted (the oscilloscope display of the trace analyzer also discourages users).

Thus, it became necessary to provide one major easily used tool to provide all the functions previously available (see Figure 2), as well as being flexible enough to meet new ones. The following tables illustrate the primary functions performed.

TABLE 1

FRONT PANEL

1) Monitoring instruction addresses as accessed
2) Monitoring data addresses as accessed
3) Starting, stopping and continuing execution
4) Resetting the status
5) Setting and clearing hardware breakpoints
6) Executing instructions external to the computer memory
7) Examining and loading memory
8) Examining and loading the accumulators
9) Examining hardware status
10) Single stepping through the program

TABLE 2

HISTOGRAM

1) Counting the number of times instructions in a given area are executed
2) Measure time spent in sections of program flow
3) Measure time spent in executive and individual user modes

TABLE 3
DEBUGGER

1) Search memory for matching words/bits
2) Have multiple, counting breakpoints
3) Format data output

TABLE 4
TRACE

The analyzer will store 1K sequences of instruction addresses, before, after or split on each side of a specified instruction execution.

The initial design of the SPA was then undertaken with these considerations. The basic design would be as shown in Figure 3 in order to simplify design and programming. A ROLM 1666 was selected for use as a control computer. Since one ROLM computer would obviously not be fast enough to monitor a second ROLM, the monitor electronics would be built using high-speed, discrete logic. This logic would include its own memory optimized to retain the state of the object machine during operation.

The control electronics were further sub-divided to allow for future changes or expansion. The hardware thus consists of the object computer and its associated peripherals, the object computer interface, the control electronics, the control computer interface, and the control computer with its associated peripherals (see Figure 4). This design permits replacing the control or object computers without redesigning the entire analyzer. Only the interface would need to be replaced.

The object computer is slightly modified by the addition of logic to produce necessary signals. These include timing signals, such as the start of an instruction, and condition signals such as the issuance of a program flow change instruction (JST, JMP, RT, etc.). These signals are sensed by monitoring a subset of the microcode instruction lines. Other important timing and condition signals, as well as the address and data bus, are available on the control panel bus. On the control panel bus are control lines that allow command of the object computer (see Figure 5). These are buffered and multiplexed to allow normal display functions of the front panel while making them available to the control electronics. This allows the computer to function normally at full speed while being monitored by the control electronics.

The control electronics are fast enough to respond asynchronously to microcode sequences within a single instruction. The major component is a number of event registers.

The SPA has eight (8) event registers. Each register may be used to define an event on the object computer. An event is said to occur if all of the "conditions" specified in the event register occur simultaneously on the object computer. Each event register may be

set up to recognize all of the following conditions on the object computer as being TRUE, FALSE or DON'T CARE.

1) Address within a specified range
2) Executive mode or user mode
3) Addressing mode (instruction fetch or data addressing)
4) Memory write operation (STORE, INCREMENT, etc.)
5) Data channel activity
6) Interrupt activity
7) Floating point processor busy
8) Program flow change instruction (JUMP, JSR, INT, etc.)
9) Carry bit
10) Overflow bit
11) Expanded memory
12) Interrupt branch mode

An event register may also set a bit to inform the other seven event registers upon the occurrence of an event. This is done with eight additional condition bits. Consequently, there are 20 condition bits associated with event register, the 12 bits described above and the eight bits indicating whether an event has already occurred on any of the eight event registers. The event registers may also be re-enabled by other event registers.

Description of the event alone is by first selecting conditions of interest with a 16-bit mask register. Then the user specifies the values of those conditions which are of interest to him (0,1) and stores these values in the condition register. If address ranges are of interest, it is necessary to set the upper and lower bounds of the address register.

Also associated with each event register is a 16-bit counter which is decremented each time an event occurs and which may be programmed to produce some action when it is decremented through zero. This counter must also be initialized.

Finally, the user must select the action he wishes to take place when the event occurs. These actions are as follows:

1) Stop the object machine
2) Activate or deactivate a special function (described later)
3) Set condition bit informing other event registers of this event

These actions are selected by control bits in a control word. To summarize, an event register consists of the following six components:

1) Condition Word (32 bits)
2) Condition Mask (32 bits)
3) Upper Instruction Address Bound (16 bits)
4) Lower Instruction Address Bound (16 bits)
5) Upper Data Address Bound (16 bits)
6) Lower Data Address Bound (16 bits)
7) Data Written (16 bits)
8) 16-Bit Auto-Decrementing Counter
9) Action Control Bits

The user can think of each event register as operating according to the following logical diagram (Figure 6).

The special functions provided are of three types. One is the timing or counting function. This provides a count of the number of instructions, or the amount of time between two selected events. The two methods of timing are provided to measure activity of the two concurrent processors, the DMA and the floating point (see Figure 7).

A second function is the histogram. This increments a local memory address representing an area of the object computer's memory (see Figure 8). The histogram count is provided by an event register. This makes it possible to count subsets, such as DMA accesses, separately.

The third major function is a trace analyzer (see Figure 9). This provides a record of addresses accessed. Operating in one of three modes, it will trace all memory accesses, instruction fetches, or program flow changes. The memory is a 1K circular buffer, wide enough to hold address and state information. The buffer will sample continuously. When triggered by an event register, it will count up to 1000 more accesses and then freeze the buffer. These special functions can be started and stopped under control of the event registers.

The interface between the control computer and the control electronics is quite simple. An I/O bus repeater in the control computer brings the signals out to the control electronics. Here, the signals are buffered and control logic decodes the appropriate registers to load (see Figure 10). The interface also loads to commands to be transferred through the control panel interface.

The user interface is a software package resident on the control computer. This allows the user to issue high level commands to set up the control electronics and the object machine, and to format the results according to his needs. This package is shown and explained below and in Figure 11.

The software for the SPA is the SPA software manager (SPASM). This consists of three major modules as shown in Figure 11. The user communicates with the system using the command language and the report generator.

The SPA control electronics will be essentially a passive device under direct control of the control computer. Consequently, the control computer will contain a comprehensive set of software modules which implement the SPA functions. The control computer will also interface directly to the user in an interactive mode of operation. The user will communicate requests to the control computer in a high-level control language and these requests will be translated into commands which will be issued to the SPA control electronics.

The SPA software manager (SPASM) will use named variables and logical constructs to coordinate a series of sequential tasks. Each task will perform one function and will consist of a series of elementary commands to the control electronics.

Certain commands may instruct the SPA control electronics to pass information describing the state of the object machine back to the control computer. For example, the control electronics may be instructed to identify certain user-specified events and to interrupt the control computer upon their occurrence. A set of utilities will exist on the control computer to handle this sort of input from the control electronics and to translate them into a form readable by the user. The SPA software will consist of the following primary components:

1) Operating System & Utilities
2) Control Electronics Interface Modules
3) SPA Software Manager
4) User Interface Modules
5) Software Modules Specific to SPA Analysis Tasks

Get User Info - This module communicates with the user to control the SPA functions.

Get Program - This module accepts a string of commands from either the console, from memory, or from a disk file.

Get New Info - The program commands are examined and any information or parameters necessry for operation are requested from the user.

Save PCM - The complete program is optionally saved in source format as a disk file.

Set-Up Control Electronics - This module manages the SPA control electronics.

Translate PGM - The program is translated into a series of machine steps. These steps include loading the control electronics registers, interrogating the control electronics to obtain information and sending the information and formats to the report module.

Load SPACE - This module loads the registers of the SPA control electronics necessary to perform the current operation.

Get Results - The SPA waits until signaled that information is available. Whatever registers or memories are then read to provide the report module with necessary information.

The commands are given through a pseudo-language.

It allows the user to write, save, and run analysis programs using a System Control and Analysis Language (SCALE). SCALE programs are a set of procedures consisting of definitions and statements. SCALE is based on "snapshots" of the state of the object machine, one taken for each instruction executed by the object machine. When the state of the object machine matches the conditions specified in one of eight event registers, an event occurs. These events activate the SCALE program.

A SCALE procedure is organized as follows.

Procedure	pname	Name of procedure
Define		Definition section.
Variable	var1,var2,...	All variables must be named. All are octal integers.
Event	event name(i)	
Condition		These conditions describe the state of the object machine for an event to occur.
condition a		
condition b		
.		
.		
.		
Action		The actions describe what effect the event will have.
action a		
action b		
.		
.		
.		
Event	event name(i)	
.		
.		
.		
Function	function	This is for special functions. The information depends upon which functions are used.
information	name(i)	
information		
.		
.		
.		
End Define		
ON event name(i)		Immediate transfer to new procedure upon event.
ON DO procedure name(x)		These are listed in order of priority.
Start		Starts object machine.
Wait		Waits for event to occur
Statement 1		Start point for program after occurrence of event
Statement 2		other than those causing immediate transfer
.		
.		
.		
Final		This terminates the procedure.
End Procedure		

A SCALE program will consist of a main procedure and Ø or more called procedures. Called procedures can cancel or redefine events and can call other procedures. Upon return from a called procedure or termination of a series of statements, the procedure will re-enter the wait state until the occurrence of another event.

Thus a SCALE program would provide the flexibility necessary to obtain full usage of the SPA hardware. A procedure could wait for a choice of events, depending on the event. The SPA could then be reconfigured, the object machine restarted, and any appropriate reports generated.

There are a number of considerations for the future. The first is to replace the ROLM 1666 as the control computer. The loading of registers is obviously something that can

be done by a microprocessor. This in turn would make possible a considerable reduction in size. A sufficient reduction in size would make possible airborne use, allowing studies to be made "on the job". A final modification would be to make this device operational in a multiprocessor configuration. This, of course, would call for major redesign.

This has been a description of a tool for analyzing the operation of a complex system. By combining fast, simple registers with a sophisticated minicomputer, we are able to operate our system without restraints and still monitor it as the instruction level.

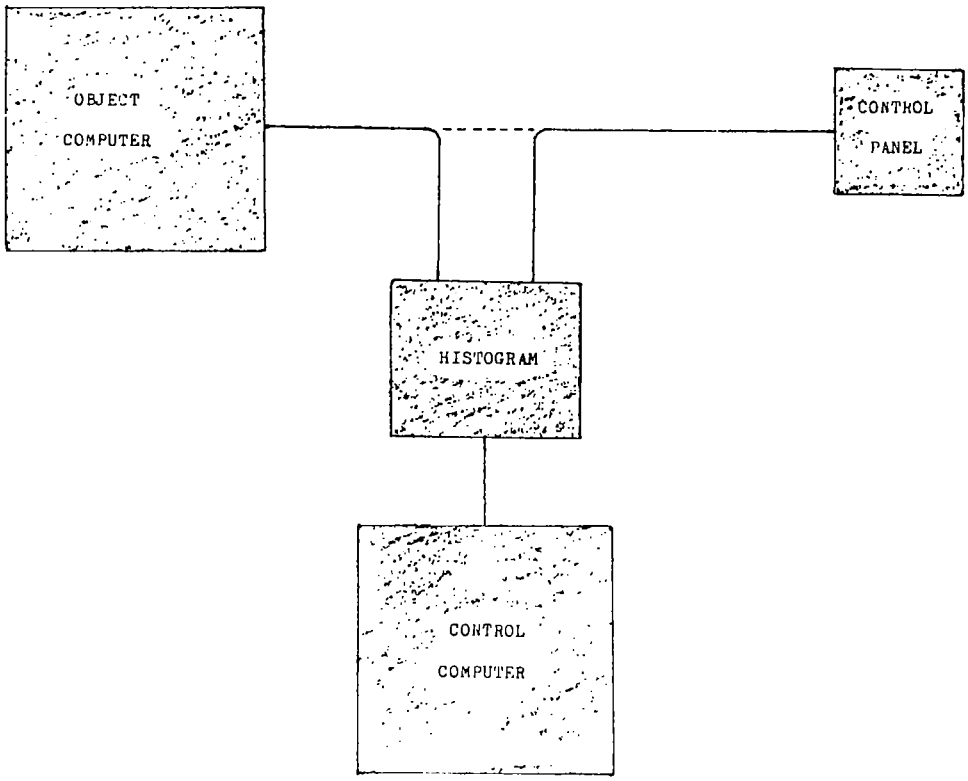


Figure 1

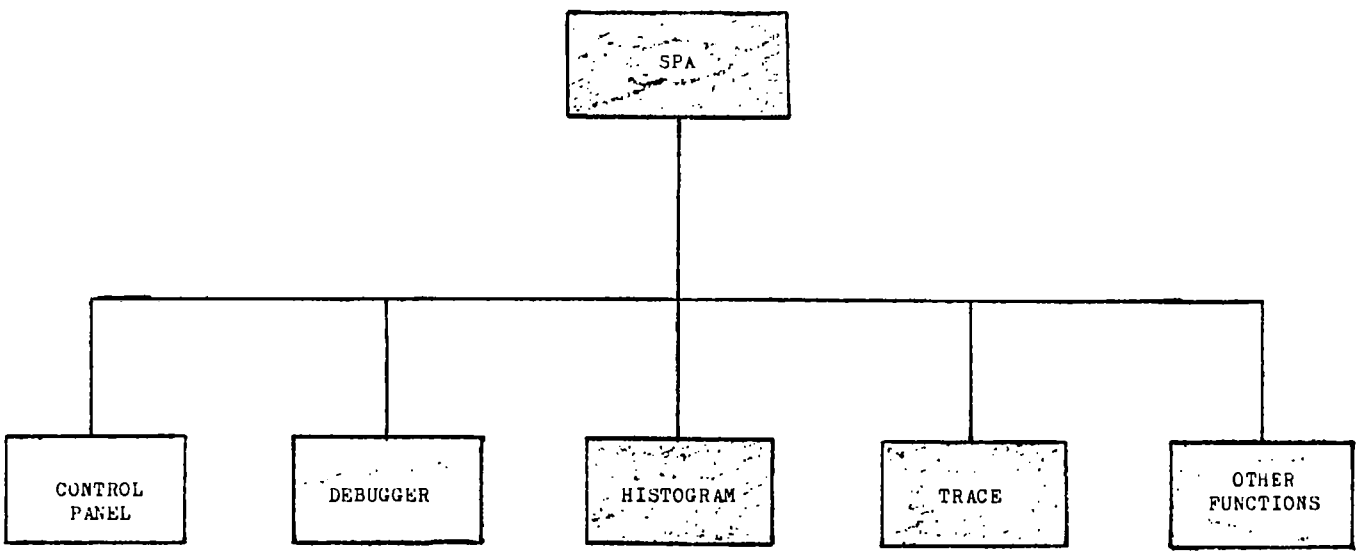


Figure 2

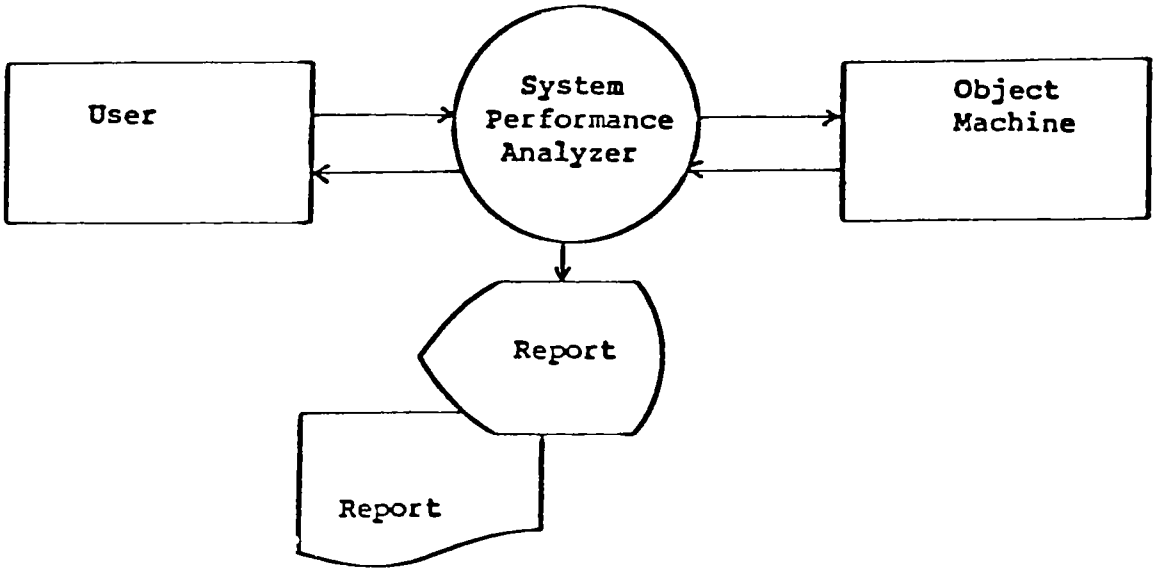


Figure 3

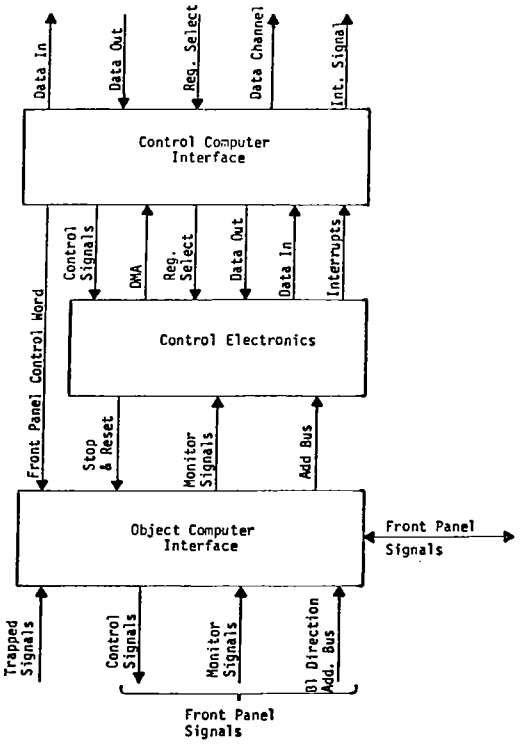


Figure 4

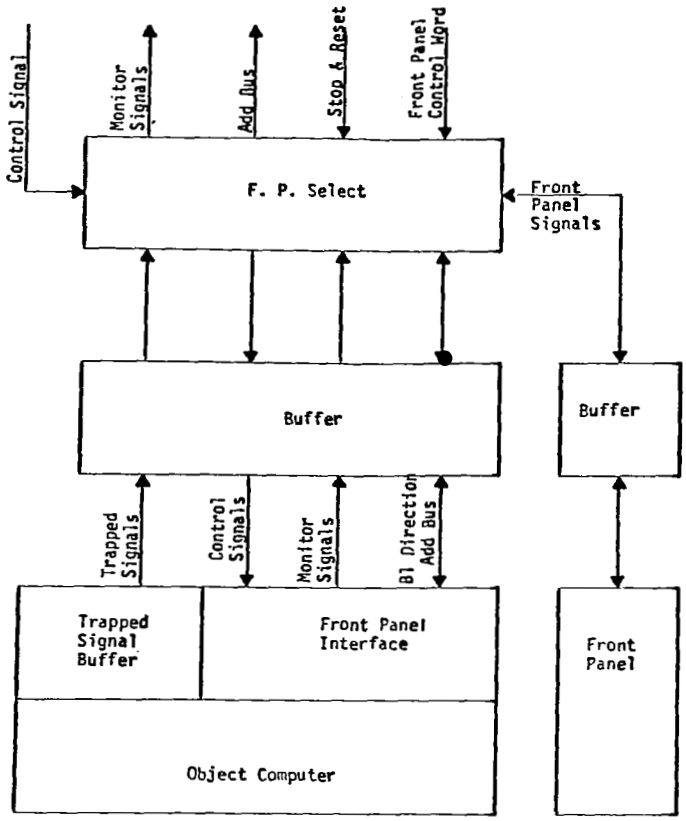


Figure 5

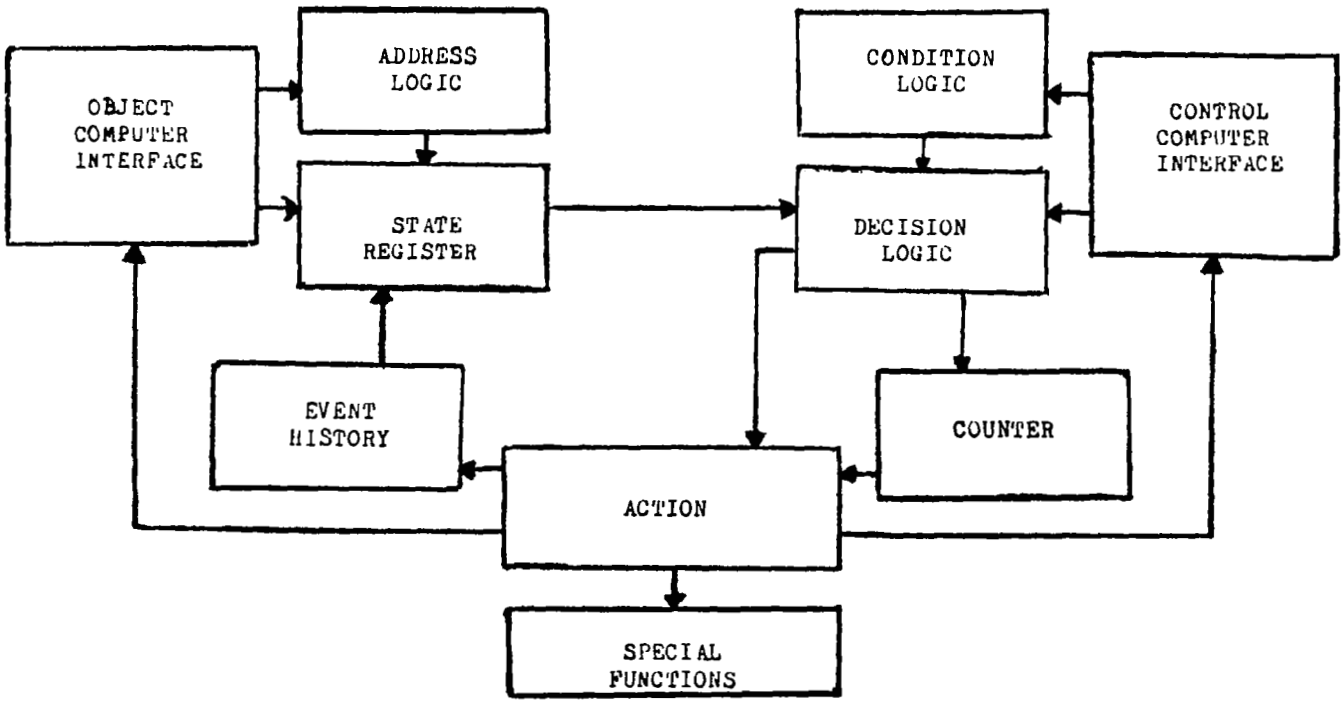


Figure 6

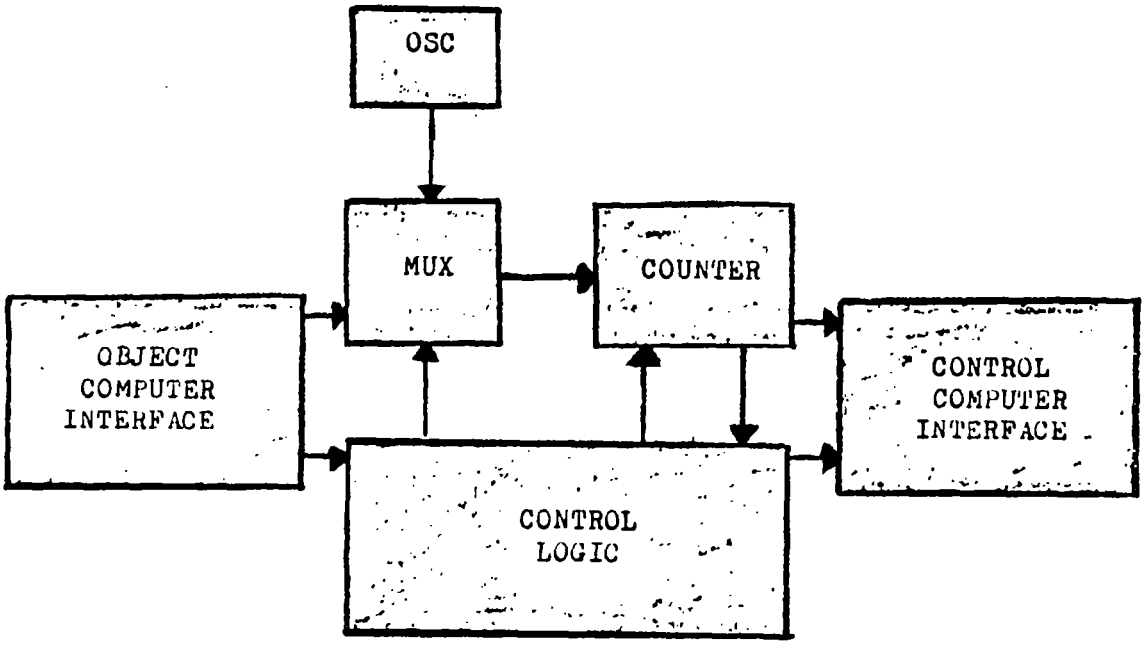


Figure 7

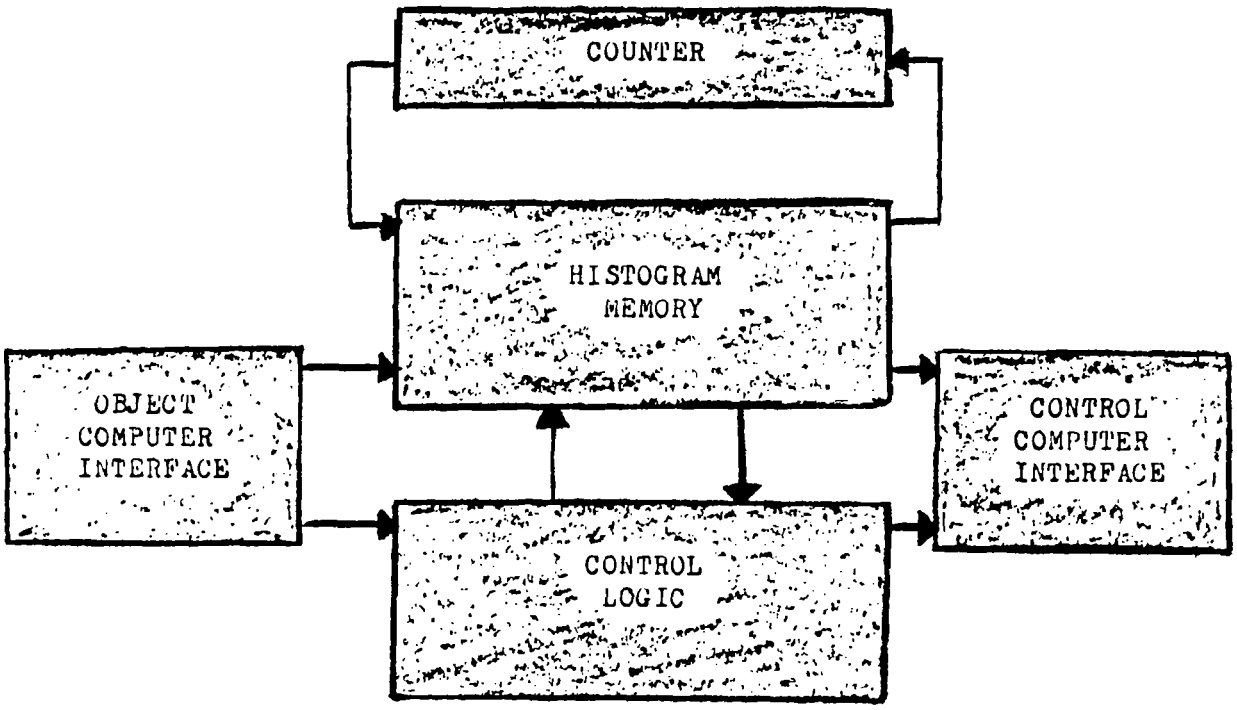


Figure 8

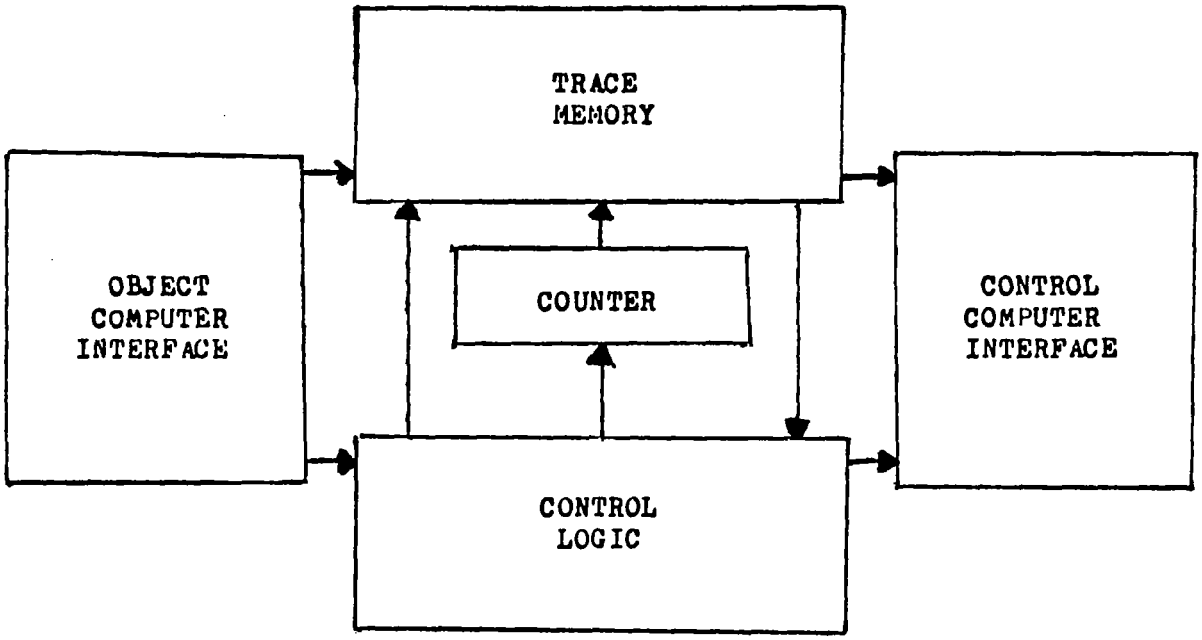


Figure 9

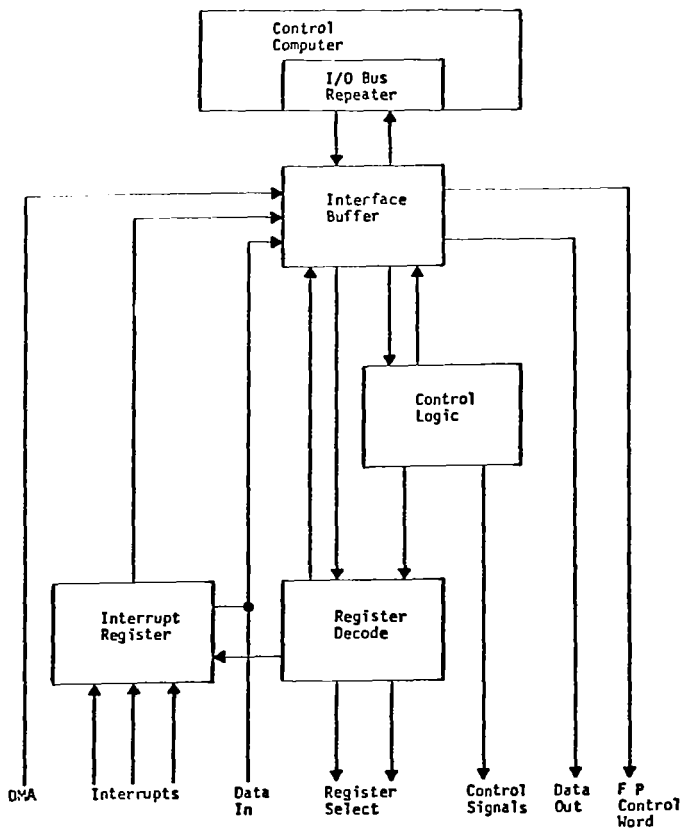


Figure 10

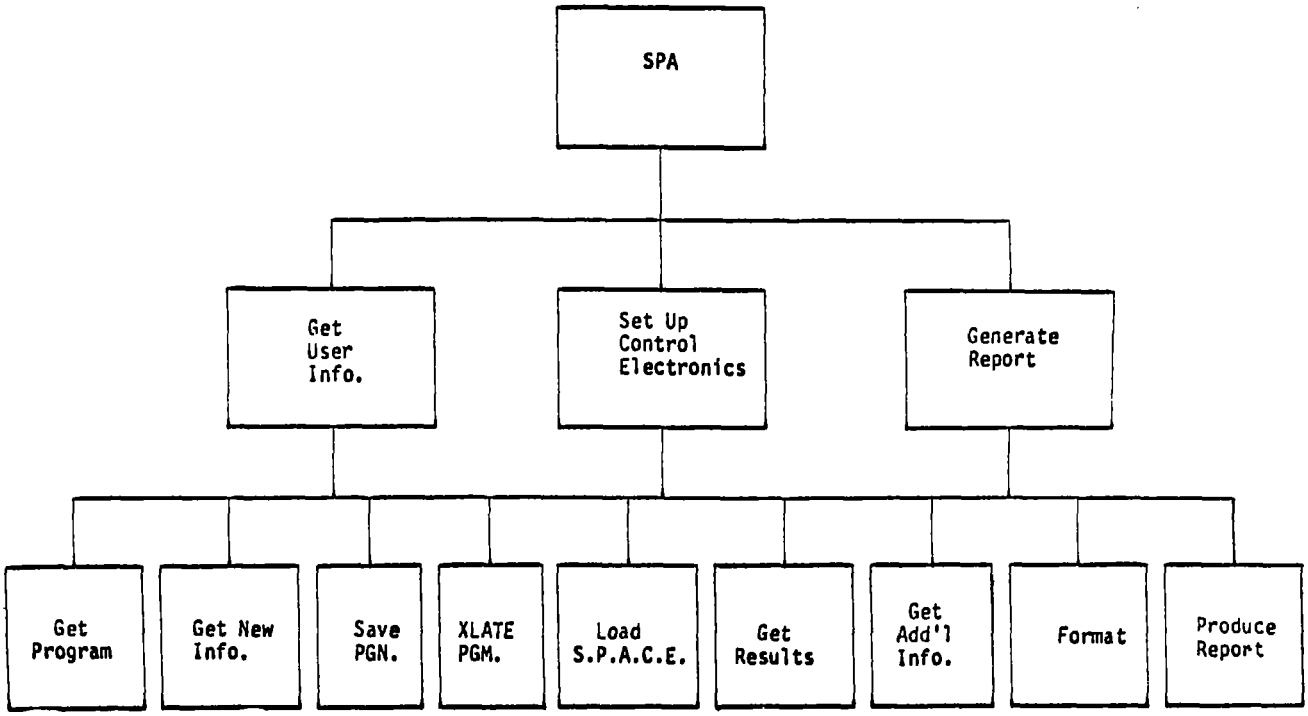

SPA
Get User Info.
Set Up Control Electronics
Generate Report
Get Program
Get New Info.
Save PGN.
XLATE PGM.
Load S.P.A.C.E.
Get Results
Get Add'l Info.
Format
Produce Report

Figure 11

ON-BOARD COMPUTER PROGRESS IN DEVELOPMENT

OF A 310 FLIGHT TESTING PROGRAM

Pierre Reau
Operations Department, Flight Test Management
Aerospatiale
France

INTRODUCTION

Aerospatiale, a French national industry, was founded on January 1, 1970, as a result of the merging of the formerly existing firms called Sud-Aviation, Nord-Aviation, and Sereb.

In 1979 the company was ranked 20th in France, employing 33 000 people. The same year its turnover figured up to 11 billion francs (about 2 billion dollars) with a 19.4% expansion rate, and 46% of its sales as exports. Its activities span four divisions:

- Aircraft
- Helicopters
- Tactical missiles
- Ballistics

Within the aircraft division, the Flight Test Management has been involved in instrumenting, designing, and conducting flight tests for the last 35 years. Work was invested in several aircraft: the Armagnac in 1949 (a long range, four-engine aircraft), the Grognard and C170 Magister (military jets), and the Caravelle in 1955 (a short range twin-engine jet). It also contributed to the flight tests performed on the Concorde in 1969 and the A 300 starting in 1972.

For the development and the final airworthiness certification of the first option of Airbus A 300, the Flight Test Group devoted 1413 hours of flight time to four aircraft over about sixteen months.

It has been estimated that both the developing and certifying of the Airbus A 310 with both available versions (Pratt and Whitney or General Electric engines) will require 1300 hours of flight time spent on five aircraft over thirteen months. Implementation of this program will entail spending 2 million dollars on on-board computer equipment and 1.5 million dollars on operational staffing as well as on a ground computer operational infrastructure.

AIRBORNE COMPUTER OBJECTIVES

The decision to install mini-computers on board the first three A 310 airplanes was made in 1979 in order to:

(a) Assure the flight safety by exercising a limit check of a given set of parameters

(b) Improve the efficiency of flight tests and allow a cost reduction by giving the crew a comprehensive look at the flight situation (this allows for dynamic decision making during the flight)

(c) Perform the tests analysis on an external basis by using the data recorded on the on-board flight tapes.

The ROLM computer was selected in 1979 from among several U.S. and European manufacturers. The ground-based system was installed in early 1980.

SIMULATION

Purposes

Prior to the writing of the technical data we felt it was first necessary to simulate the on-board computer using a SEL 86 Computer (Systems Engineering Laboratories). The simulation design consisted of:

(a) Optimizing the sequencing of the different phases to minimize computations and eliminate duplications. This allows a fast reduction of memory requirements and improves the utilization of CPU time

(b) Estimating the program size given the complexity of the task

(c) Using a SEL 86, the characteristics of which are well understood, to set a baseline to be used as a criterion for the selection of an on-board computer

(d) Optimizing the call and scheduling of tasks to simplify the flight observer activities

(e) Providing an analyzing department with user-oriented output formats

Equipment Used

Simulation has been performed on a SEL 86 (fig. 1), involving mainly:

- A central unit with a 64-K-word (32-bit) memory and a console
- A 900 lines/minute capacity printer (132 c/line)
- A 400 cards/minute capacity reader
- Two magnetic tape units (1.14 m/sec, 40.64 bits/m)
- One 100-megabyte disk unit as a back-up to the Real Time Monitor System
- One reading unit for the flight tapes recorded on board the TSS and A 300

Simulated Programs Characteristics

The processing programs were written in assembly language although an optimized SEL Fortran was available. They performed computations and provided flight data output results. Not all possible programs were simulated, only those directly related to the following flight phases: take-off/landing, cruise, and rate of climb.

All acquisition phases for the selection of samples to be processed, the conversion into industrial units, and the data transfer were timed using the SEL 86 internal clock.

Conclusions Drawn On Simulation

The on-board computer simulation, implemented in the SEL-86, required the adjustment of several SEL-86 system parameters. These adjustments were primarily made to the simulation program's data transfer phases onto computer-compatible tapes and output phases to oscillographic recorders. Before these adjustments, the simulation program had been run sequentially at the maximum speed possible. The simulation has thus disclosed the following points:

(1) It is better to cut down the flight tape reading speed and perform several real time treatments simultaneously in lieu of a serial treatment

(2) What had been simulated would be realistic even if the expected mini-computer was up to three times slower provided a real-time-oriented system had been used with a fast memory capacity, sufficient mass memory, and multiprocessor and multiprogram function capabilities, connected to several peripherals like printer, plotter, and monitor

SHORT DESCRIPTION OF THE AIRBORNE EQUIPMENT ON THE A 310

Flight tests will be conducted with five aircraft, three of them fully equipped with airborne computers. The equipment includes the following (fig. 2):

- 800 to 1400 measurement chains, or numerical bus delivering analog and digital data
- Two digital acquisition systems with a sampling rate capability ranging from 1 to 128 samples/second; each acquisition system delivers a message of 32 000 12-bit words per second
- The ROLM system is designed for real time processing and visualization of data on screens (this package will be further described) and is linked to one of the two acquisition systems
- An analog display onto a trace recorder
- Digital display panels for warning, etc.
- Closed-circuit television panels
- Two flight tape transports for 2.5-cm-wide (1"-wide) magnetic tapes providing a six-hour recording of numerical messages coming from the acquisition system and used for replay on the ground computer
- One analog acquisition system with corresponding recording units for measurement chains having a 2-KHz band pass or more

CUB DESCRIPTION

The on-board Universal Calculator consists of two parts:

(1) The ROLM 1666 system

(2) The visualization system using an AFIGRAF CRT connected to the ROLM 1666

1666 Airborne System

Each airborne system (fig. 3) includes:

- Two ROLM 1666 central units, one equipped with a 32-K memory (CPU 1), the other with a 64-K memory (CPU 2)
- Two 2150 I/O chassis. One is used to input the flight data obtained from the acquisition system. All peripherals and the warning transmitter are connected to the other unit (CPU 2)
- Two ROLM 1648 control panels
- One floppy disk system using 3 floppy disks (1 master 3385 and 2 slave 3386's)
- One Versatec 7200 printer plotter (printing speed 1000 lines/min, 132 characters/line)
- One ZIP 30 typewriter used as a console

The CPU's are linked by a ROLM model 3550 Multiprocessor Communications Adapter.

Both the data transfer towards the visualization system screens and the input of the data provided by the airborne acquisition system are performed by direct memory access using 3564 Data Channel Controller interfaces. A 3549 System Interrupt card allows proper timing between the data input and its subsequent use in the ROLM program.

Airborne Visualization Kits

This system (fig. 3) is designed by a French company (La Compagnie des Signaux et d'Entreprises Electriques), a subsidiary G3S INFODIF group. Each airborne outfit includes:

- One screen processor equipped with 4000 words of 16-bit memory and a card that generates interrupts, i.e. end-of-image interrupt
- One screen demultiplexer which is used to separate the two images, which are both stored in the single processor's memory
- Two Hewlett Packard 1311A CRT's (21 cm x 21 cm with a definition of 1024 x 1024 points).
- One display analyzer capable of printing hard copy from one CRT (it should be noted that the monitor generating a hard copy is frozen for two seconds only)

·One Tektronix 4632 hard copy unit

The screens are refreshed at a rate of 50 times/second, thus allowing for a visualization of motion displays and still offering the features of a continuous phenomenon. The data transfer between the ROLM memory and the display is achieved at the maximum permissible speed allowed by the data channel controller.

Ground System

This system is used for the program implementation (fig. 4). In addition to the airborne system we have:

·One 3341 moving head disk system
·One 3367 commercial MTU
·Two 3302 video terminals
·One alphanumeric keyboard for the AFIGRAF System
·One light pen

The mag tape is used to communicate between the CUB and the other ground computers, SEL 86 and SEL 32/77. The disk unit is used to support the RMX/RDOS system.

Flight Informations Input

The airborne acquisition system (fig. 5) provides two digital messages each consisting of 32 000 12-bit words/sec. The long cycle is made up of 16 mean cycles and each mean cycle is made up of 8 short cycles.

The information is stored in the 32-K-word memory, each 12 bits being stored in a 16-bit word using one of two 3564 Data Channel Controllers, each 1/8 of a second in flip-flop mode (called a double cycle).

One interruption at the start of the long cycle is sent to the central unit 1 while the first word of the long cycle is being stored. An interruption of the double mean cycle is effected while the first word of the double mean cycle enters. Interruptions are generated by an interface 3549.

Switching of messages on the 3564 interfaces and generation of interrupt signals on the 3549 are handled by a special module. We preferred this solution to any other, using all possibilities offered by the DCC. It permits a better control of the validity of the messages and increases data flow at the DCC entry, modifying, if necessary, time intervals of commutation.

PROGRAM SPECIFICATION

Programs are developed on the ground equipment, under RMX/RDOS. In-flight, they are run under RTOS (fig. 6). We define two distinct categories:

(1) Permanent programs: They are permanently stored, whether active or not. These programs are loaded automatically when the on-board system is started.

(2) Specific programs: Each is relevant to one type of test. Their executions are temporary and are called up by the flying observer. They are activated and stopped either by the observer or automatically.

Permanent Programs

(a) Sample choice: This program extracts in CPU 1 the samples required by all the analysis programs and converts them into industrial units. To keep computation time to a minimum, each sample is only extracted once per double mean cycle and can then be used, if required, by all the analysis programs.

(b) General performance: This program is designed to compute specific parameters like altitude, speed, and the Mach number necessary to most of the other programs (number of inputs about 20).

(c) Limit check: Parameters that are subject to a limit check are grouped into basic lists to which complementary lists can be associated. Should a parameter exceed its limits, a warning is displayed to the crew. The parameter values of both the basic and complementary lists can be output on the printer on request. The list of parameters to be monitored can be modified throughout the flight.

(d) Flight profile: This program stores the flight profile information at regular intervals on floppy disks. A listing of the recorded values is extracted after the flight, on the ground equipment. This information is used by the data processing team. Throughout the flight the observer can request this data to be displayed on the printer.

(e) Data visualization on screens: Two screens are used to display simulated cockpit instruments, i.e. circular, linear, and digital indicators (fig. 7). On one of the screens general parameters are given, showing the aircraft's attitude, power settings, weight, flaps, etc. The remaining screen is used to display specific parameters relevant to the test in progress. Throughout the flight the observer may, at his discretion, change the format of this last screen.

Specific Programs

These are temporary programs, applicable to different types of tests such as cruising performances, quality of flight, etc. The method of calling, starting, execution, and stopping is a function of the program. The ZIP 30 typewriter is used to control the operations of the program, the modification of the basic parameters list used in limit check, and the screen format.

Analysis Frequency and Program Execution Priority

The analysis frequency is not necessarily the same for all the programs. The smaller this period, the higher the program priority. For instance, programs like sample extraction and general performances computation are processed 8 times/second, whereas the flight profile is only processed every 10 seconds.

TRIALS AWAY FROM TOULOUSE

During flight test trials away from Toulouse, the airborne CUB system will be used to replay the flight tapes on the ground. It will then be possible to analyze some of the data after the flight.

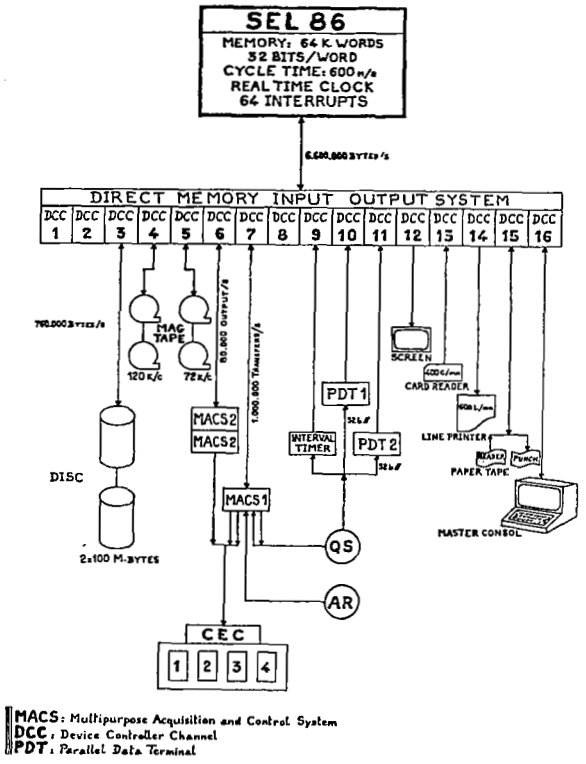


Figure 1

SCHEMATIC OF THE FLIGHT INSTALLATION

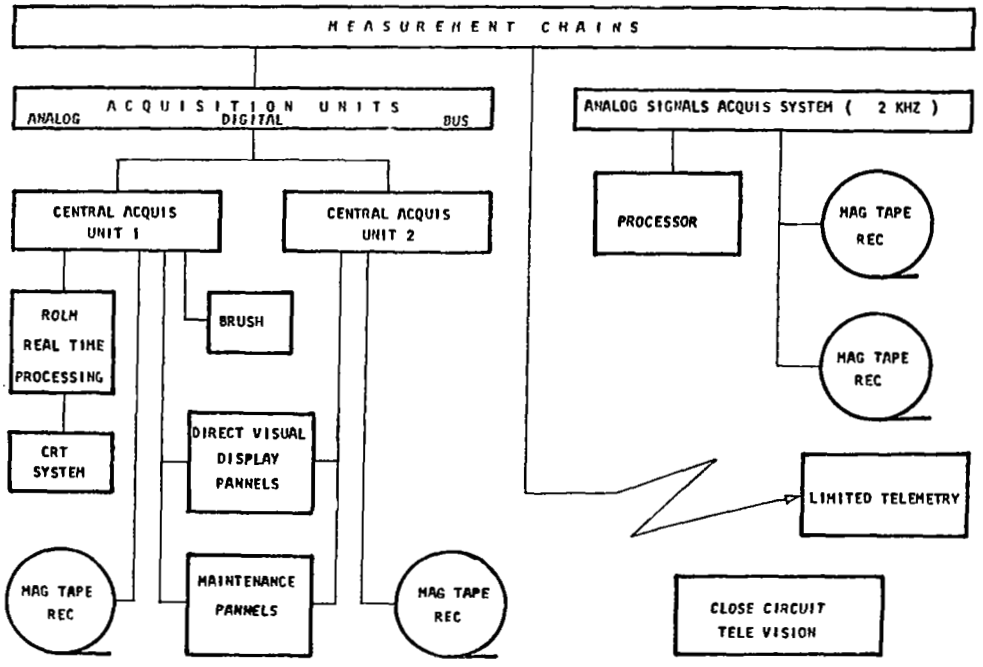


Figure 2

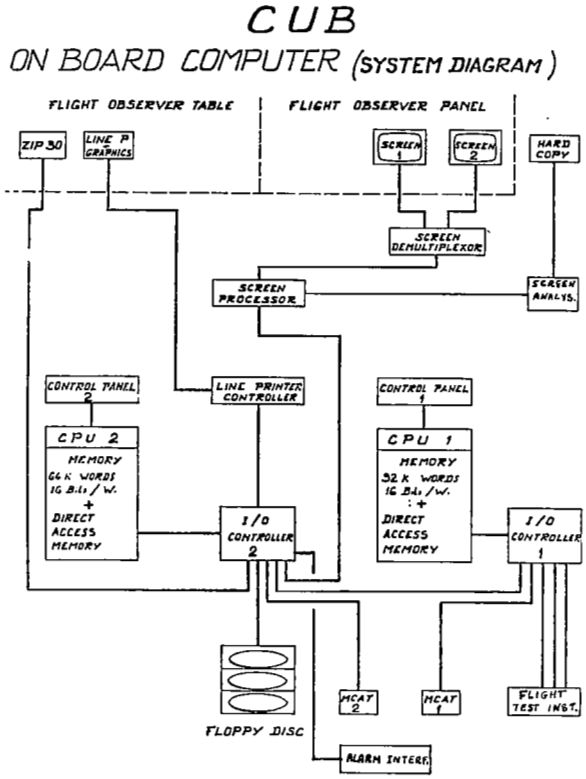


Figure 3

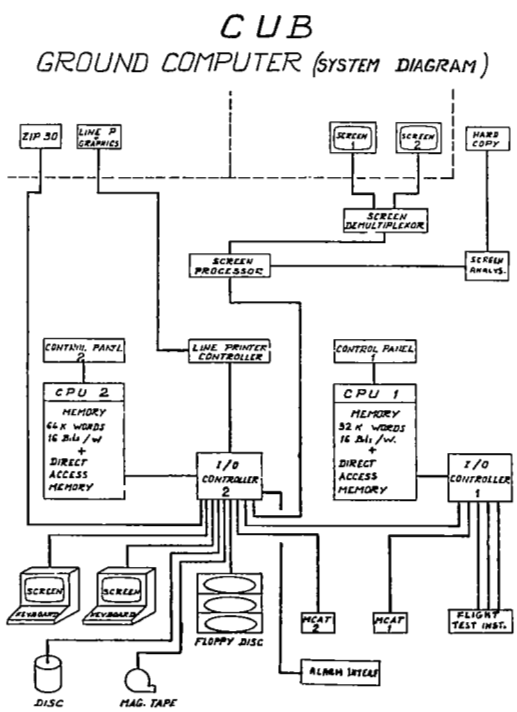


Figure 4

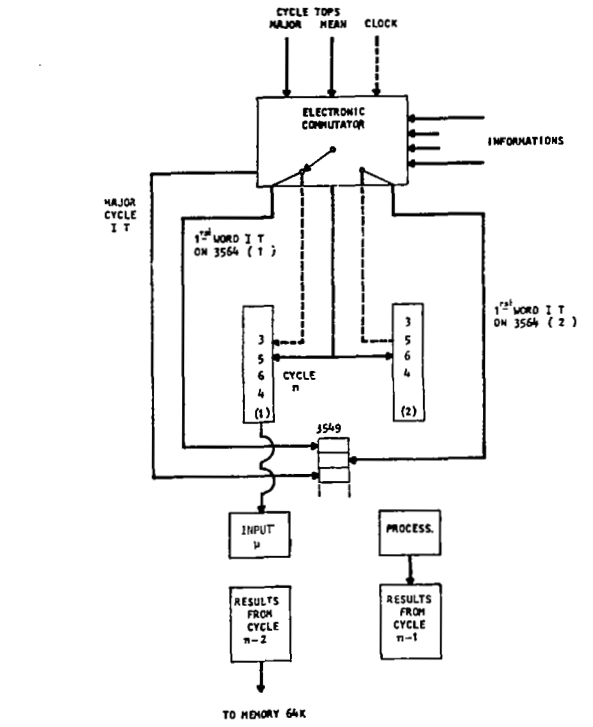


Figure 5

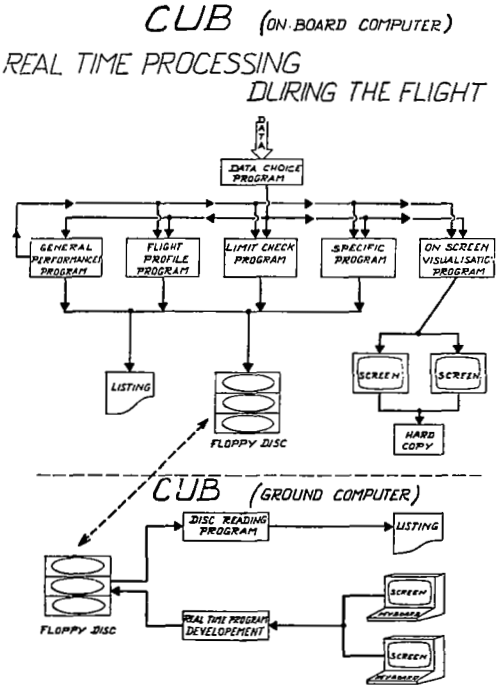


Figure 6

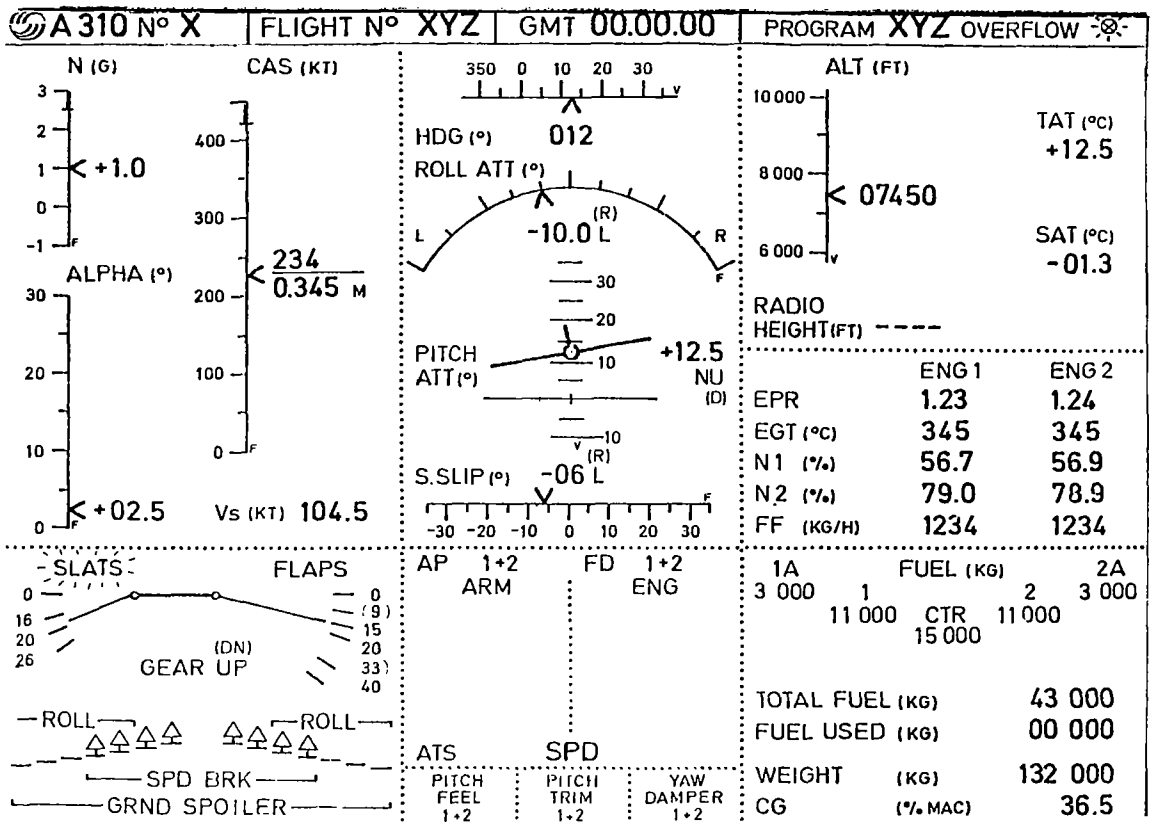

A 310 N° X
FLIGHT N° XYZ
GMT 00.00.00
PROGRAM XYZ OVERFLOW
N (G)
CAS (KT)
+1.0
234
0.345 M
ALPHA (°)
+02.5
Vs (KT) 104.5
HDG (°) 012
ROLL ATT (°)
-10.0 L
PITCH ATT (°)
+12.5
NU
S.SLIP (°) -06 L
ALT (FT)
07450
TAT (°C)
+12.5
SAT (°C)
-01.3
RADIO HEIGHT (FT) ----
ENG 1
ENG 2
EPR 1.23 1.24
EGT (°C) 345 345
N1 (%) 56.7 56.9
N2 (%) 79.0 78.9
FF (KG/H) 1234 1234
SLATS
FLAPS
(DN)
GEAR UP
ROLL
SPD BRK
GRND SPOILER
AP 1+2
ARM
FD 1+2
ENG
ATS
SPD
PITCH FEEL 1+2
PITCH TRIM 1+2
YAW DAMPER 1+2
FUEL (KG)
1A 3 000
1 11 000
CTR 15 000
2 11 000
2A 3 000
TOTAL FUEL (KG) 43 000
FUEL USED (KG) 00 000
WEIGHT (KG) 132 000
CG (% MAC) 36.5

Figure 7

NEW STARTS

IN RESEARCH AND DEVELOPMENT 1982

Joseph Grosson
Executive Director for Acquisition
Naval Material Command, Navy Department
Washington, D.C.

ABSTRACT

This paper outlines, in slide form, some areas of new U.S. Navy research and development utilizing minicomputers.

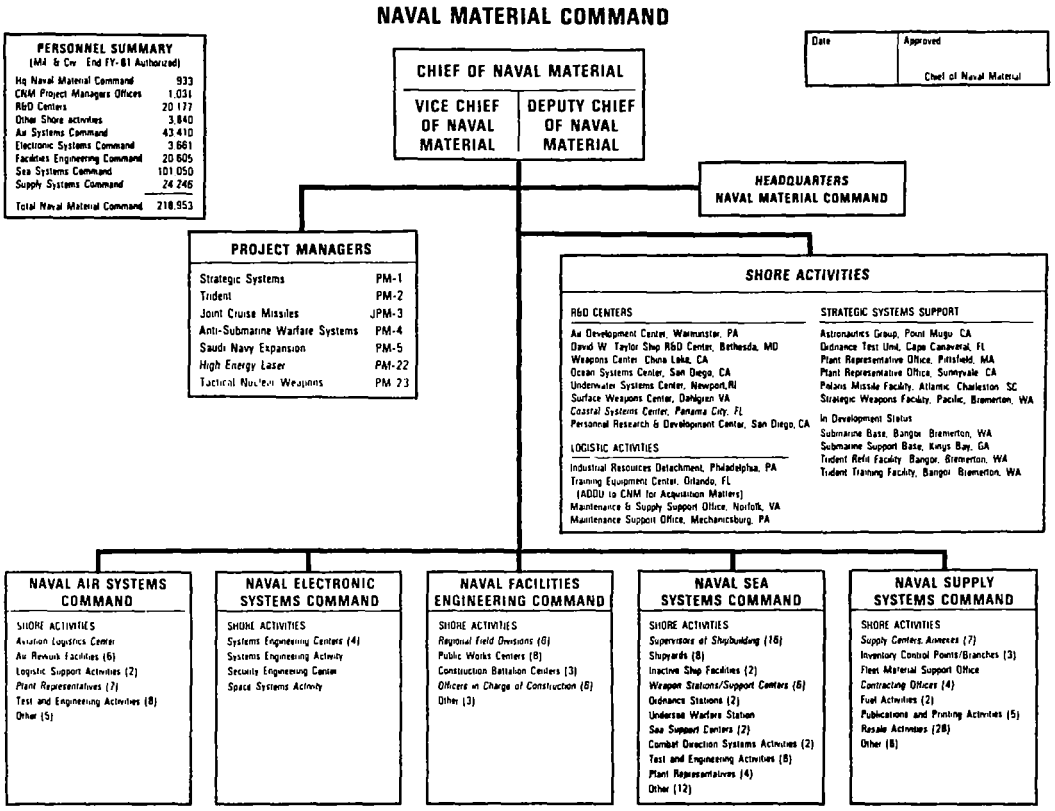


Figure 1

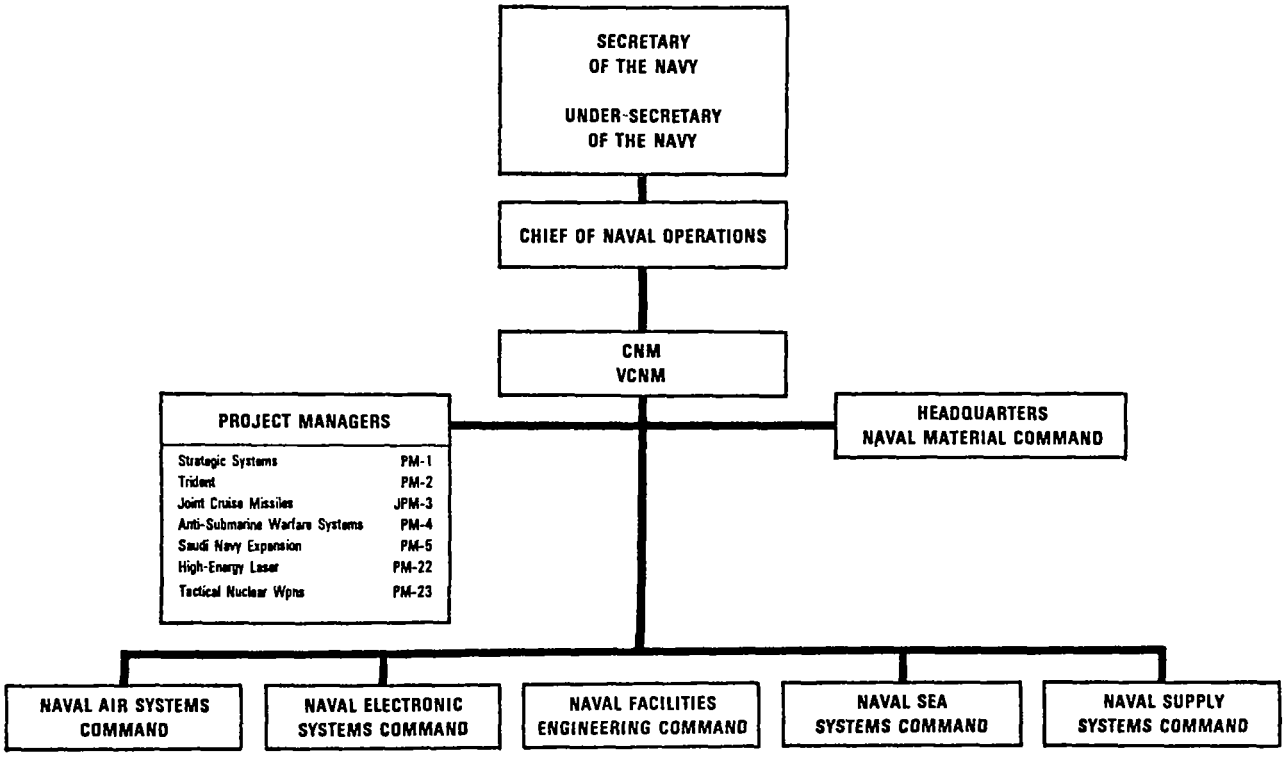


Figure 2

CNM ORGANIZATION

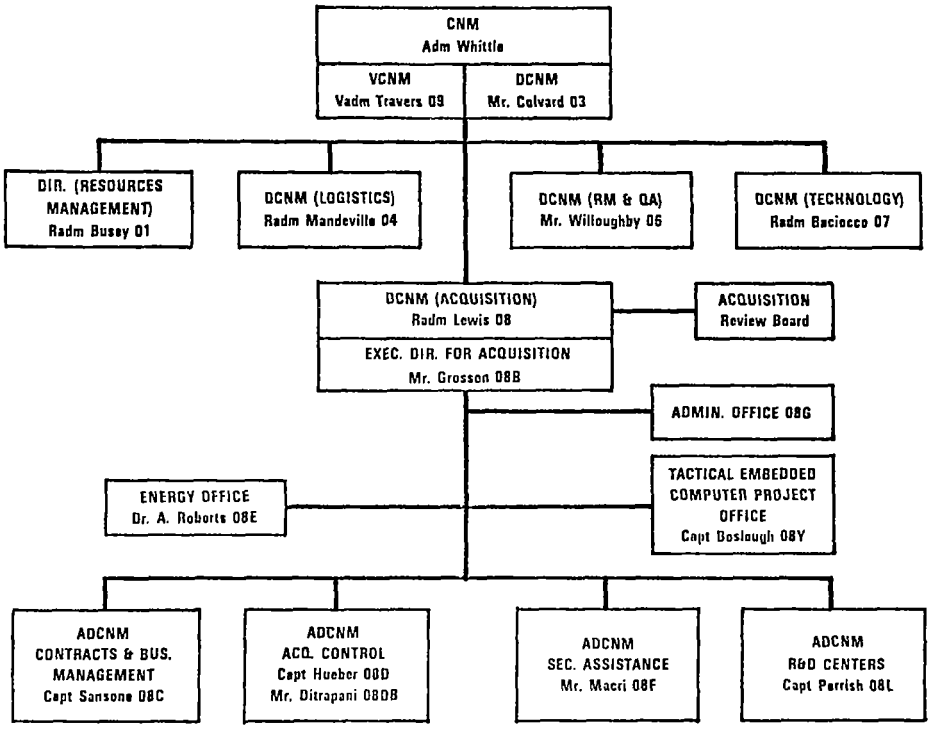


Figure 3

ALL NAVY R&D SUMMARY
PRES. BUDGET Jan 81 FYDP

	FY 1980	FY 1981	FY 1982
RESEARCH	$214,941	$241,813	$285,777
EXPLORATORY DEVELOPMENT	$396,207	$459,648	$515,527
ADVANCED DEVELOPMENT	$1,032,867	$1,244,114	$1,692,597
ENGINEERING DEVELOPMENT	$1,851,932	$1,758,849	$2,021,111
MANAGEMENT & SUPPORT	$424,944	$488,641	$595,344
OPERATIONAL DEVELOPMENT	$642,379	$702,044	$755,932
TOTALS	$4,563,270	$4,895,109	$5,866,288

Figure 4

NAVY RDT&E PROGRAMS
FY 82

6.1	RESEARCH	$285M
6.2	EXPLORATORY DEVELOPMENT	$515M
6.3	ADVANCED DEVELOPMENT	$1,692M
6.4	ENGINEERING DEVELOPMENT	$2,021M
6.5	MANAGEMENT SUPPORT	$595M
6.6	OPERATIONAL SYSTEMS SUPPORT	$755M

Figure 5

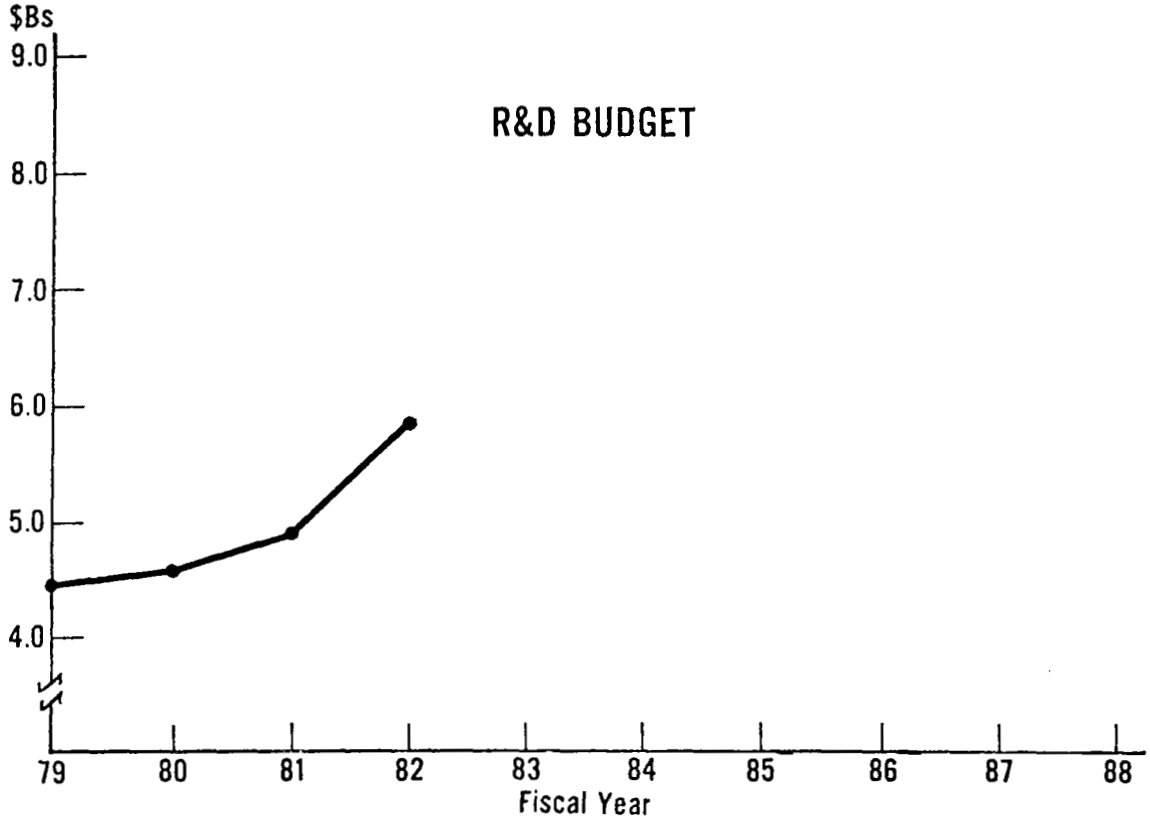


Figure 6

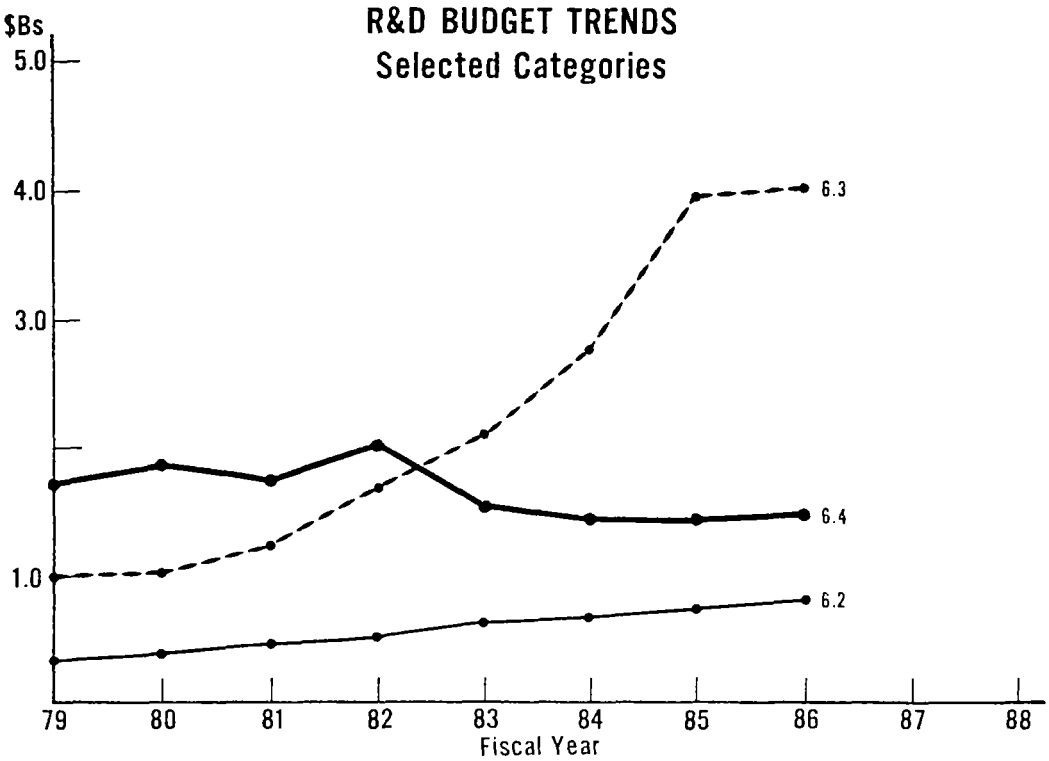


Figure 7

NEW START SUMMARY

FY 1982

Category	P.E.	New P.E.	Proj	New Proj
6.2 EXPLORATORY	22	1	246	16
6.3 ADVANCED	141	13	343	53
6.4 ENGINEERING	98	6	284	28

Figure 8

FY 1982 OSD/OMB SUBMISSION
OCTOBER FYDP 18 SEP '80
6.2 PROGRAMS

PROGRAM TOTAL (BUDGET MINIMUM)	$ 549.695M
PROGRAM ELEMENTS	22
NEW P.E.s	2
PROJECTS	246
NEW PROJECTS (FY 82)	16

Figure 9

NEW STARTS - PROGRAM ELEMENTS (6.2)

PE	TITLE	YEAR	AMOUNT
62735N	HIGH-ENERGY LASER TECHNOLOGY	FY 1981	$33.023
62768N	DIRECTED ENERGY TECHNOLOGY	FY 1982	$10.173

Figure 10

FY 82 NEW STARTS - PROJECTS (6.2)

P.E.	PROJECT	TITLE
62757N	RF57525	HUMAN INFO PROCESS IN C^2
62758N	MF58523	PERS PROTECTION & SURVIVAL
62758N	RF58523	PERS PROTECTION & SURVIVAL
62758N	MF58527	CASUALTY CARE
62758N	MF58528	BIOMED EFFECT ON PERS PERF
62758N	RF58528	BIOMED EFFECT ON PERS PERF
62760N	WF60531	FLEET LOGISTICS READINESS TECH
62763N	ZF63500	USMC PERS RESOURCE MGMT
62765N	WF65573	PYROTECH TECH & COMPONENTS
62768N	RF68311	SYS INVST DIRECT ENERGY WPN
62768N	RF68342	ADV CHEM & EXCIMR LASR TECH
62768N	WF68342	FREE ELECTRON LASER TECH
62768N	SF68343	CHARGED PARTICLE BEAM TECH
62768N	WF68344	HI POWER MICROWAVE TECH
62768N	RF68345	ADV LASER OPTIC TECH
62768N	WF68345	PULSED POWER TECH

Figure 11

ACTIVE ADJUNCT TO UNDERSEA SURVEILLANCE

ALTERNATIVE TO PASSIVE SURVEILLANCE TO COUNTERACT REDUCED RADIATED SIGNATURES.

A FIVE-ELEMENT SYSTEM DEMONSTRATION WILL BE CONDUCTED IN UTILIZING HYDROACOUSTIC SOURCES IN CONJUNCTION WITH A MID-FREQUENCY TOWED ARRAY DEPLOYMENT FROM THE SAME SHIP.

ELEX/NOSC $2,500K (FY 82)

Figure 12

ADVANCED ELECTRICAL PROPULSION SYSTEMS

ADVANCED ELECTRICAL PROPULSION SYSTEMS OFFER REDUCED COST AND MORE FLEXIBLE ARRANGEMENTS FOR SHIP AND SUBMARINE CONSTRUCTION. RECENT ANALYSES HAVE SHOWN THE POTENTIAL FOR A 20% REDUCTION IN OVERALL SURFACE SHIP DISPLACEMENTS FOR COMPARABLE RANGE/PAYLOADS.

SEA/DTNSRDC $4,700K (FY 82)

Figure 13

HIGH-CAPACITY LOW-VOLUME BATTERIES

A HIGH-RATE LITHIUM THIONYL CHLORIDE BATTERY IS A CANDIDATE POWER SOURCE FOR TORPEDO PROPULSION. IT HAS POTENTIAL FOR HIGH POWER, WAKELESS, SILVER-FREE ENERGY SOURCE.

SEA/NOSC/NUSC $800K (FY 82)

Figure 14

MARINE CORPS WEAPONRY IMPROVEMENT

IMPROVE BATTERY FIRE POWER BY ADAPTING ZUNI ROCKETS TO A GROUND-BASED ROLE AND DEVELOPING LIGHTWEIGHT CARRIAGES FOR LARGE-CALIBER GUNS. AN IMPROVED HAWK SYSTEM IS ALSO UNDER DEVELOPMENT AS A HIGHLY MOBILE AIR DEFENSE SYSTEM CAPABLE OF ENGAGING MULTIPLE SIMULTANEOUS TARGETS ON THE BATTLE FIELDS OF THE 1980's AND 90's.

NSWC - DAHLGREN $4,780K (FY 82)

Figure 15

- IMPROVED METAL MATRIX COMPOSITES
  - SHIPBOARD ANTENNAS - RADAR PEDESTALS
  - MISSILES AND AIRFRAME STRUCTURES
- LASER - ASSISTED METALWORKING
  - AUTOMATED WELDING OF SHIP STRUCTURES
  - CORROSION-/EROSION-RESISTANT SURFACES - GEARS/BEARINGS
- RAPID SOLIDIFICATION PROCESSING OF ALLOYS
  - HOMOGENEOUS CORROSION-RESISTANT ALLOYS
  - SHIP PROPELLER AND IMPELLER BLADES

Figure 16

IMPROVED METAL MATRIX COMPOSITES

METAL MATRIX COMPOSITES, WHICH PROVIDE LOW DENSITY, STIFFNESS, STRENGTH AND THERMAL STABILITY ARE BEING DEVELOPED FOR APPLICATION TO KINETIC PENETRATORS, SHIPBOARD ANTENNAS - RADAR PEDESTALS, TACTICAL MISSILES AND AIRFRAME STRUCTURES.

SEA/NSWC $3,500K (FY 82)

Figure 17

LASER-ASSISTED METALWORKING

BY COMBINING THE DEMONSTRATED HIGH-SPEED PROCESSING CAPABILITY OF THE LASER BEAM WITH AUTOMATED ADAPTIVE CONTROLS, LASER-ASSISTED METALWORKING IS BEING DEVELOPED FOR APPLICATION TO SHIP, SUBMARINE AND AIRCRAFT PRODUCTION AND REPAIR.

SEA/NRL $150K (FY 82)

Figure 18

RAPID SOLIDIFICATION PROCESSING OF ALLOYS

CHARACTERIZE THE UNIQUE MICROSTRUCTURE AND PROPERTIES RESULTING FROM RAPID SOLIDIFICATION PROCESSING OF METAL ALLOYS. IMPACTS:

- GAS TURBINE ENGINES
- CORROSION-RESISTANT ALLOYS AS REPLACEMENT FOR STAINLESS STEEL
- NEW STRUCTURAL ALUMINUM ALLOYS FOR INCREASED TEMPERATURE APPLICATIONS

AIR/NRL/DTNSRDC $850K (FY 82)

Figure 19

OSD/OMB SUBMISSION

OCTOBER FYDP 18 SEP 80

6.3 PROGRAMS

PROGRAM TOTAL (BUDGET MINIMUM)	**$ 1,487.977M**
PROGRAM ELEMENTS	**141**
NEW P.E.s	**26**
PROJECTS	**343**
NEW PROJECTS (FY 82)	**53**

Figure 20

FY 82 NEW STARTS - PROJECTS (6.3)

P.E.	PROJECT	TITLE
63207N	W1399-OS	NOSS
63216N	W1401SL	HELO AIRCREW SURVIVABILITY
63217N	WO885SL	MOD AVIONICS PKG
63217N	WO892CC	INFO HAND SYS
63262N	WO592-SL	A/C & ORDNANCE SAFETY
63267N	W1253-AA	NATO FUTURE IDENTI SYS
63308N	WO440-AA	RAMJET MISSILE TECH
63313N	WO302SH	IR ATTACK WEAPON
63369N	W1446-TW	MRASM (IIR)
63371N	R1452-SB	GEO SAT
63506N	SO225-AS	SURFACE SHIP TORP DEF
63520N	X1286-CC	NAVY FUTURE COMM SYS
63523N	S1332-SL	SWATH
63524N	S1440-AS	EMSP
63533N	S1417-SL	SHPBD CORROSION CONT
63534N	SO308-SH	SES
63536N	SO854-AA	SOJS
63564N	S1357-AS	FFX
63573N	S1314-SL	ELECTRIC DRIVE
63589N	S1448-SL	NON-AEGIS RADAR DEV
63589N	S1449-SL	LIGHTWEIGHT AEGIS
63589N	S1450-SL	COMBAT SYSTEM INTEGRATION
63589N	S1451-SL	LIGHTWEIGHT SONAR
63635N	C1295-AW	ARTY DIFIS
63707N	Z1383-PN	CIVIAN PERSONNEL ISSUES
63707N	Z1385-PN	COMPUTERIZED ADAPTIVE TEST
63710N	RO126-PN	OPERATIONAL DECISION AIDS
63710N	T1393-PN	MICROFILM TECH FOR RECORDS
63710N	W1230-PN	DESIGN FOR MAINTAIN
63710N	Z1170-PN	HUM PROC AUTO DATA BASE
63710N	Z1392-PN	PERFORMANCE ENHANCEMENT
63720N	Z1382-PN	FUNCTIONAL CONTEXT TRN
63720N	Z1388-PN	LOW COST MICRO COMP SYS
63730M	CO066-CC	NON COMM ECM SYS
63720M	CO937-SL	MOBILE EW SUPP SYS
63730M	C1296-CC	ALL SOURCE IMG PROC
63730M	C1421-CC	LTWT BATTLEFIELD SURV RADAR
63730M	C1422-CC	LTWT SEIS/ ACOU PASS DEV
63731M	CO064-CC	MAR INTEG PERS SYS
63733N	W1208-PN	COMP GEN IMAGERY FOR SIM
63733N	W1209-PN	DYNAMIC SCENE VIS DISPLAY
63733N	W1389-PN	VTOL VTRS LASER DISPLAY
63733N	W1390-PN	MULTI-SPECTRAL IMAGE SYS
63733N	W1391-PN	HELMET - MOUNTED DISPLAY
63763N	X1319-OS	TACT SURVEILLANCE SYS
63784N	X0756-OS	LTWT UNDERSEA SENS COMP
63785N	RO119-OS	SURVEIL ENVIRON ACOUS SPT
63785N	RO120-SN	TAC ASW ENVIR ACOUS SPT
63785N	WO646-TW	ABN ELECTRO/OPTICAL C/M
63785N	WO659-TW	E/O GUIDED WPNS C/M TEST

Figure 21

NATIONAL OCEANIC SATELLITE SYSTEM (NOSS)

PE: 63207N SUB PROJ: W1399-0S

DESCRIPTION - A TRI-AGENCY PROGRAM (NAVY, NASA, NOAA) TO DEVELOP A CAPABILITY TO UTILIZE SATELLITE-BORNE SENSORS, GROUND PROCESSING AND SPACECRAFT CONTROL CENTERS, DATA ARCHIVAL AND PRODUCT DISTRIBUTION SYSTEMS TO MEET NAVY FLEET REQUIREMENTS FOR GLOBAL REAL-TIME OCEAN SURFACE DATA (SST, WINDS, ICE, WAVES, TOPOGRAPHY)

COGNIZANT ACTIVITY

NAVAIRSYSCOM (AIR-370G)

FIRST YEAR NAVY FY-82 FUNDING

$(M) 46.122

Figure 22

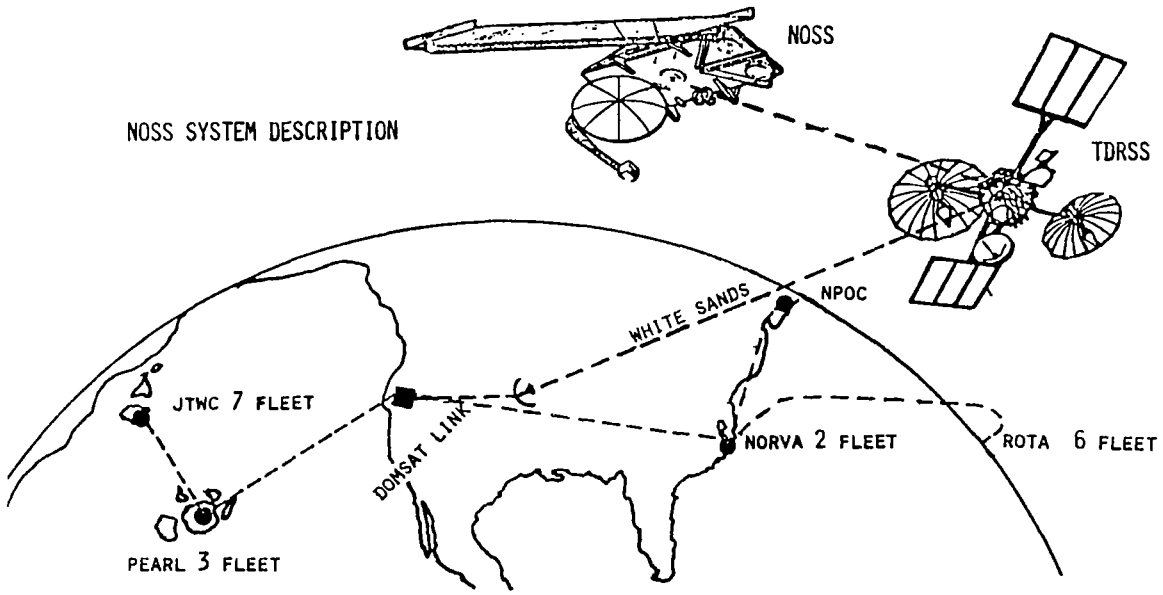


Figure 23

INFORMATION HANDLING SYSTEM

P.E. 63217N: ADV. AIRCRAFT SUBSYSTEMS

PROJ. NO. W0892-CC

PROJ. TITLE: INFORMATION HANDLING SYSTEM DEV. & EVAL.

THE INFORMATION-HANDLING SYSTEM PROJECT WILL PROVIDE REAL-TIME, DIGITAL SYSTEM ARCHITECTURES FOR INTEGRATED CORE AND MISSION AVIONICS, VEHICLE ELECTRONIC SYSTEMS, AND WEAPONS MANAGEMENT SYSTEMS FOR POST-1985 NAVAL AIRCRAFT PLATFORMS. FAULT-TOLERANCE AND RECONFIGURABILITY CONCEPTS WILL BE INCLUDED TO PROVIDE A HIGH DEGREE OF AVAILABILITY OF AIRCRAFT FOR BOTH CARRIER AND OTHER AIR-CAPABLE SHIP APPLICATIONS. STATE-OF-THE-ART, AUTOMATED DECISION AIDS (ARTIFICAL INTELLIGENCE) TECHNOLOGY WILL BE EMPLOYED TO REDUCE AIR CREW WORKLOAD.

IOC – 1987

BUDGET

FY82	FY83	FY84	FY85	FY86	FY87
1.0	1.8	3.0	2.6	2.9	(3.9)

Figure 24

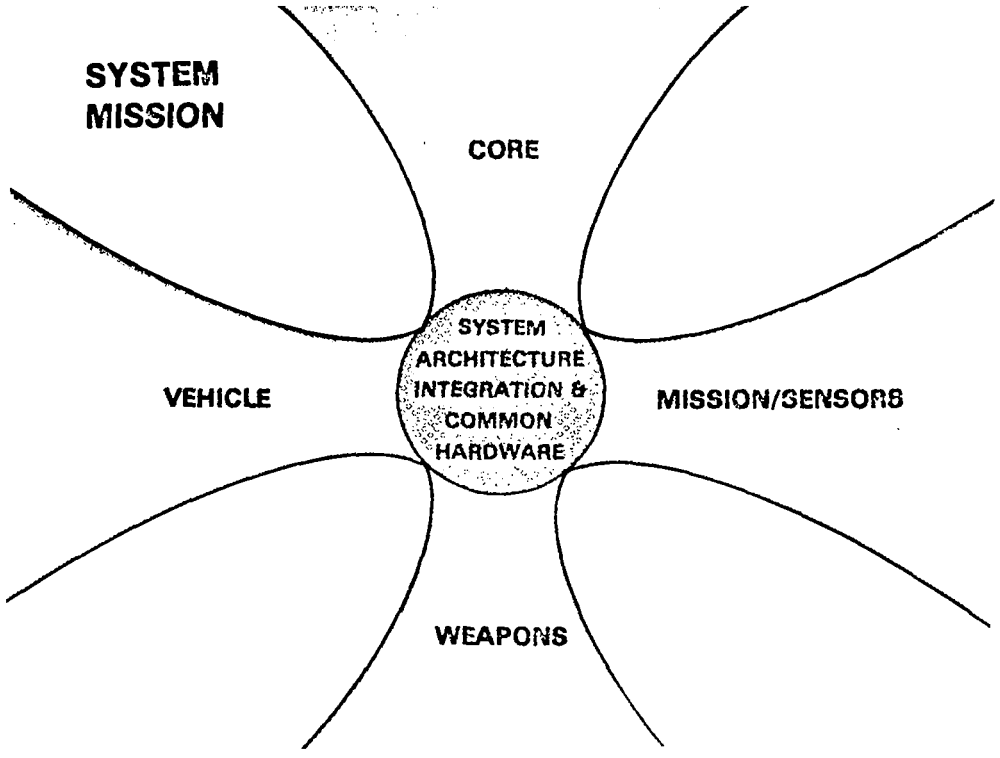


Figure 25

NATO FUTURE IDENTIFICATION SYSTEM (FIS)

P.E. 63267N

PROJ NO. W1253AA

PROJ TITLE: NATO FUTURE IDENT. SYSTEM

THIS PROJECT IS ALSO KNOWN AS NATO IDENTIFICATION SYSTEM (NIS) OR COMBAT IDENTIFICATION SYSTEM (CIS).

THIS PROJECT PROVIDES NAVY FUNDING TECHNICAL SUPPORT TO A DOD-INITIATED TRI-SERVICE PROGRAM WHICH IS ASSIGNED TO THE AIR FORCE AS LEAD SERVICE (AERONAUTICAL SYSTEMS DIVISION/XRQI, COL. BOLEN, WPAFB, O.).

THE NATO FIS WILL BE A NEW GENERATION NATO-COMPATIBLE IFF SYSTEM WHICH IS TO REPLACE THE MK X, XII IFF WHICH IS USED IN CIVIL AND MILITARY AIRCRAFT AND SHIPS WORLDWIDE. A STANAG, STANDARD NATO AGREEMENT, IS BEING STAFFED IN ALL NATO COUNTRIES WITH VARIOUS TECHNICAL APPROACHES BEING CONSIDERED. EVENTUALLY A "SIGNALS IN SPACE" STANDARD WILL BE ADOPTED, WITH EACH COUNTRY FREE TO BUILD ITS OWN EQUIPMENT. AN IOC OF 1990 IS HOPED FOR.

COGNIZANCE

NAVAIR (MR. THYBERG)

NRL (MR. VERONDA)

BUDGET	FY81	82	83	84
$M	-0-	2.5	6.9	8.0

Figure 26

GEODETIC/GEOPHYSICAL SATELLITE (GEOSAT)

PROGRAM ELEMENT NUMBER
63371N

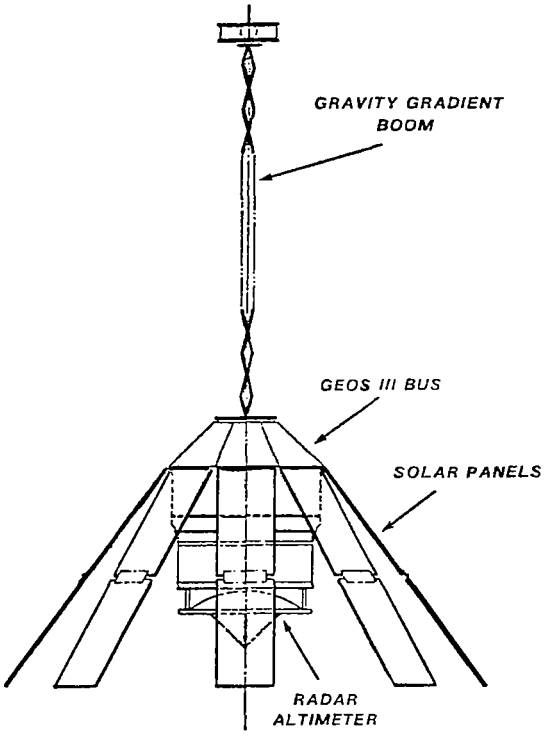
SUB PROJECT NUMBER
R1452-SB

PROJECT DESCRIPTIONS: THE GEOSAT MISSION IS TO FLY A DUPLICATE OF THE SEASAT-A RADAR ALTIMETER IN FY84. THE GEOSAT PROGRAM WILL COMMENCE IN FY82 WITH THE JOHNS HOPKINS UNIVERSITY/APPLIED PHYSICS LABORATORY DESIGNING AND FABRICATING THE SPACECRAFT. MISSION OPERATION IS SCHEDULED FOR AN 18-MONTH CONTINUOUS PERIOD. THE DATA OBTAINED WILL ALLOW IMPROVEMENTS IN THE EARTH GRAVITATIONAL MODELS IN SUPPORT OF ADVANCED SLBM SYSTEMS.

COGNIZANT ACTIVITY:
OFFICE OF NAVAL RESEARCH
CODE 464

FIRST YEAR (FY 82) FUNDING
$16,188M

Figure 27



GEOSAT

Figure 28

SURFACE SHIP TORPEDO DEFENSE

PROGRAM ELEMENT NO. 63506 **PROJECT NO. S0225**

THE SSTD PROJECT WILL PROVIDE CAPABILITIES NEEDED TO INCREASE SHIP SURVIVABILITY IN A TORPEDO THREAT ENVIRONMENT. SUBSYSTEMS WILL ADD THE ABILITY TO: DETECT AND CLASSIFY TORPEDOES, DEPLOY COUNTERMEASURES AGAINST ACOUSTIC HOMING TORPEDOES, AND DETECT ACTIVE ACOUSTIC EMISSIONS. IOC IS PLANNED FOR 1991/ 2. THE POTENTIAL MARKET FOR AT LEAST PART OF THE SYSTEM WOULD INCLUDE ALL NAVY AND COAST GUARD SURFACE SHIPS.

NAVAL SEA SYSTEMS COMMAND

FY 82 FUNDING
$2.1M

Figure 29

SSTD SUBSYSTEMS

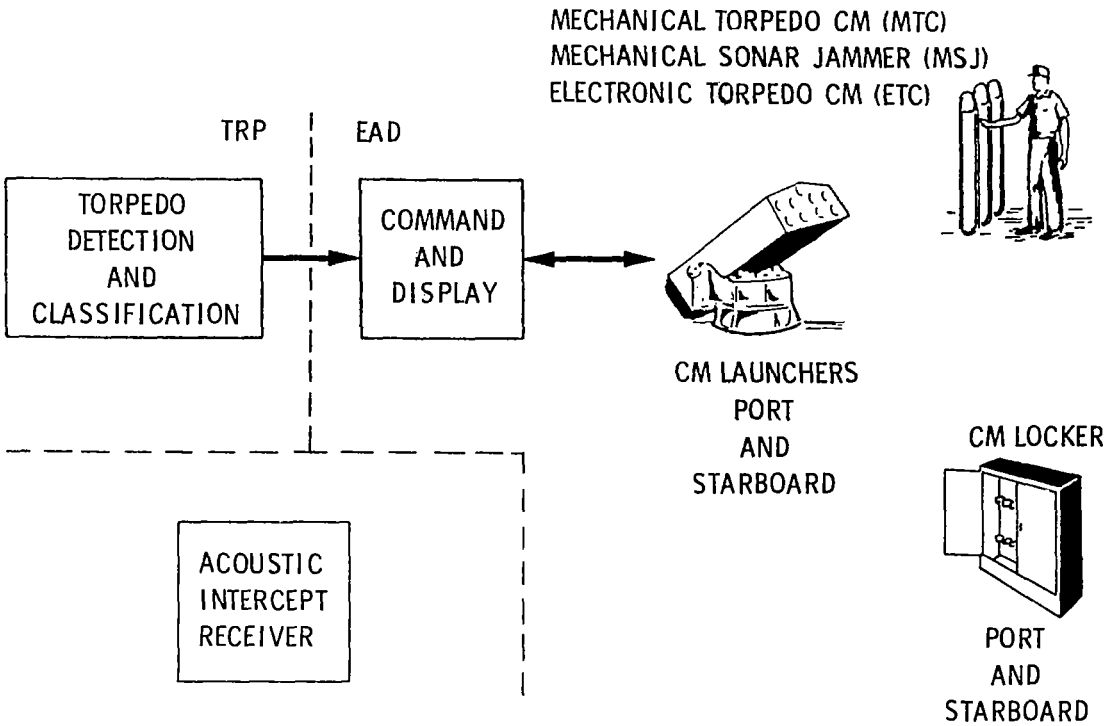


Figure 30

Shipboard Corrosion Control

Program Element Number - 63513N

Sub Project Number - S1417-SL

This Project Will Develop Production Processes to Improve Life Cycle Costs of Ships Components Through Improved Corrosion and Wear Characteristics. The Processes Will Have a Technology Base in 6.2 Area/Industry R&D.

Cognizant Activity - NAVSEA 05R15

First Year (FY 82) Funding -$505K

FY	82	83	84	85	86
$ K	505	2450	2555	2762	2764

Figure 31

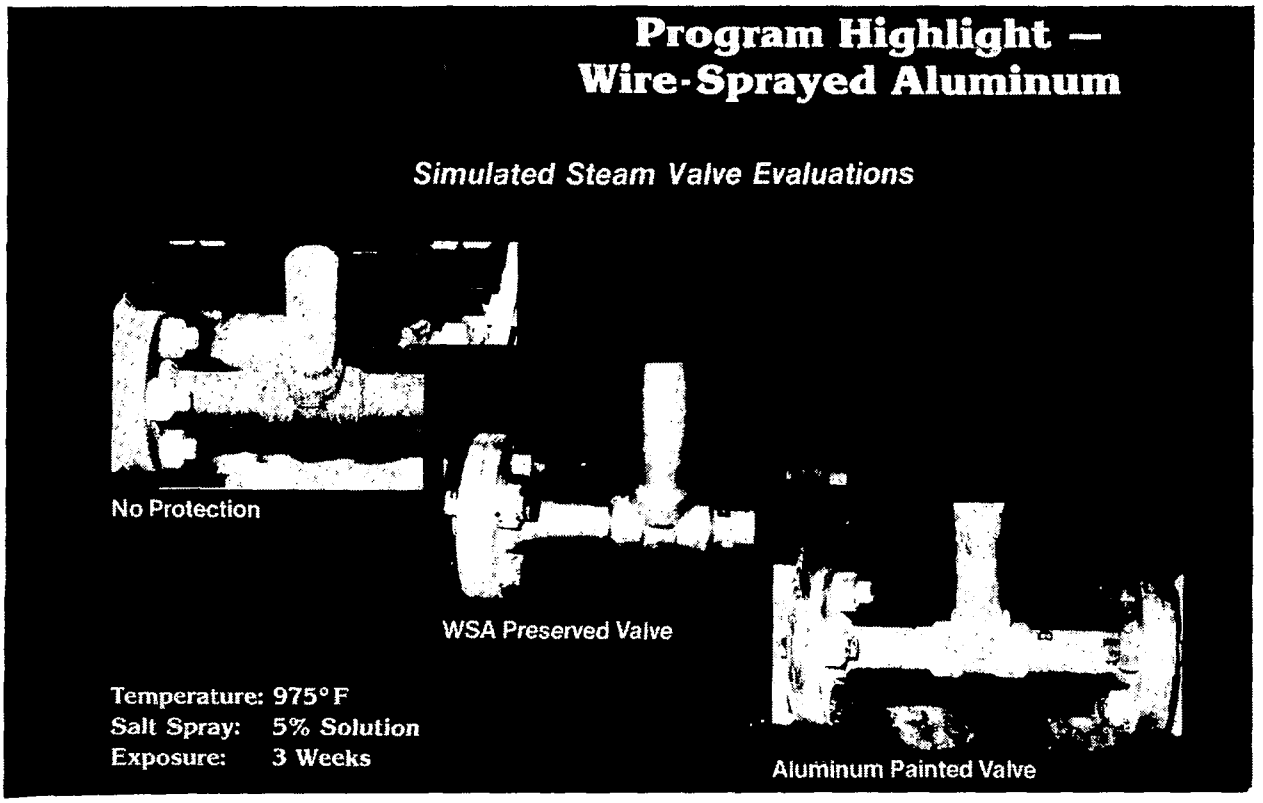


Figure 32 (Note: $975^{O}F = 524^{O}C$).

NAVY FUTURE COMMUNICATIONS SYSTEMS

P. E. 63520N PROJECT X1286-CC

PROJECT DESCRIPTIONS:

-NONSATELLITE RELAY
-LINE OF SIGHT/EXTENDED LINE OF SIGHT AJ COMMS
-DIRECTIVE SHIPBOARD ANTENNA SYSTEM
-NON-HF CHANNEL EVALUATION AND SELECTION SYSTEM
-TASK FORCE/SHIP-SHORE NETWORKS
-AFLOAT/ASHORE MODULAR RADIOS

COGNIZANT ACTIVITY	FIRST YEAR (FY82) FUNDING
NAVAL ELECTRONIC SYSTEMS COMMAND (ELEX 310)	$2.6M

Figure 33

NONSATELLITE RELAY

- OBJECTIVE
  - PROVIDE EXTENDED LOS COMMUNICATIONS
  - ADD TO THE MIX OF MISSION CAPABILITIES
  - CONSERVE AIRCRAFT RESOURCES FOR PRIME MISSION
  - PROVIDE ELOS UHF COMMUNICATIONS IN A HOSTILE ELECTRONIC ENVIRONMENT
- RATIONALE
  - DECREASE TASK FORCE RELIANCE ON HF FOR OTH
  - INCREASE COMMUNICATIONS SURVIVABILITY
  - DECREASE RELIANCE ON FIXED COMMUNICATIONS

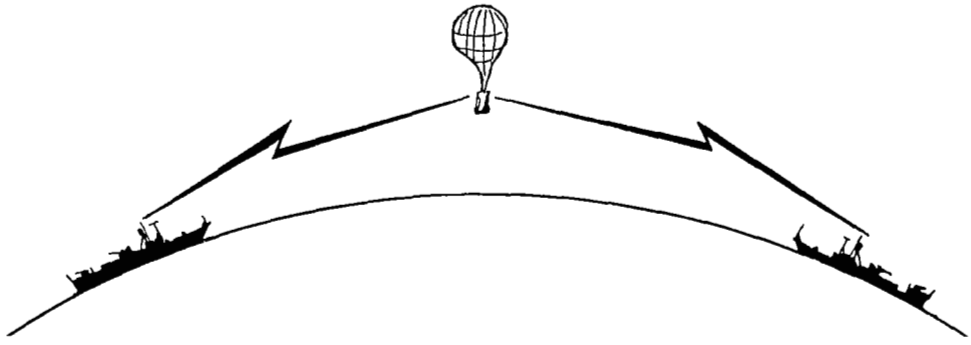


Figure 34

LINE OF SIGHT/EXTENDED LINE OF SIGHT AJ COMMUNICATIONS

- OBJECTIVE
  - PROVIDE NEW LOS/ELOS UHF/VHF AJ CAPABILITY FOR VOICE/DATA
  - ACCOMMODATE NTDS
- RATIONALE
  - REQUIREMENT TO PROVIDE FOR TASK FORCE LOS/ELOS AJ COMMUNICATIONS

Figure 35

DIRECTIVE SHIPBOARD ANTENNA SYSTEM

- OBJECTIVE
  - PROVIDE FOR STEERABLE ANTENNA WITH FOLLOWING CHARACTERISTICS FOR AJ/LPI COMMUNICATIONS
    - SELECTABLE BEAM WIDTH
    - ROTATABLE ON HORIZONTAL AND VERTICAL AXES
    - MULTIPLE BAND
- RATIONALE
  - PROVIDE FOR TASK FORCE AJ/LPI CAPABILITY
  - REDUCE HIGH SHIPBOARD RFI

Figure 36

NON-HF CHANNEL EVALUATION AND SELECTION SYSTEM

- OBJECTIVE
  - EMPLOY CES CONCEPT IN UHF/VHF/SHF BANDS
- RATIONALE
  - DESIRE TO REDUCE O&M EXPENSES
  - REQUIREMENTS TO PROVIDE FOR
    - IMPROVED CHANNEL UTILIZATION AND CIRCUIT SELECTION
    - JAMMING DETECTION

Figure 37

TASK FORCE/SHIP-SHORE NETWORKS

- OBJECTIVE
  - EMPLOY MULTIPLE ACCESS TECHNIQUES TO INTRA-TASK FORCE AND SHIP-SHORE COMMUNICATIONS (HF/UHF)
- RATIONALE
  - PROVIDE INCREASED COMMUNICATIONS PERFORMANCE WITH LIMITED RESOURCES
  - DESIRE TO REDUCE O&M EXPENSES ASHORE AND AFLOAT
  - OPPORTUNITY TO REDUCE ACQUISITION COSTS

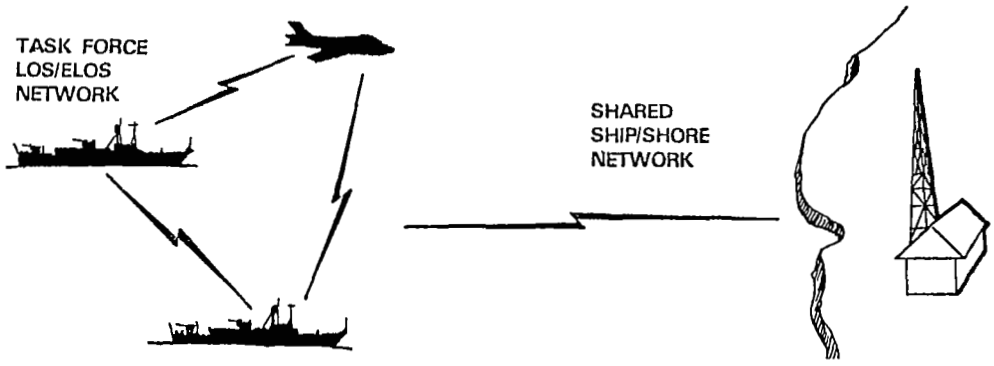


Figure 38

AFLOAT/ASHORE MODULAR RADIOS

- OBJECTIVE
  - DEVELOP A MODULAR FAMILY OF RADIO COMPONENTS TO SATISFY THE FOLLOWING CHARACTERISTICS
    - APPLICABLE TO VLF THROUGH EHF BANDS
    - PROVISION FOR AJ/LPI MODULES
    - FLEXIBILITY TO ACCOMMODATE NEW TECHNOLOGIES
- RATIONALE
  - OBSOLETE RADIO REPLACEMENT
  - MATURING VLSI/VHSIC TECHNOLOGIES
  - REDUCED O&M EXPENSES
  - REQUIREMENTS TO PROVIDE FOR
    - INTRA-TASK FORCE/LONG-HAUL AJ/LPI CAPABILITIES
    - EQUIPMENT FLEXIBILITY

Figure 39

ENHANCED MODULAR SIGNAL PROCESSOR

PE 63524N PROJECT S1440-AS

* NEXT GENERATION NAVY STANDARD SIGNAL PROCESSOR
* ORDER OF MAGNITUDE IMPROVEMENT OVER AN/UYS-1
* DEVELOPED AS AN INTEGRAL MEMBER OF NAVY STANDARD TACTICAL EMBEDDED COMPUTER FAMILY
* INITIAL DEVELOPMENT UNDER THE SUBMARINE ADVANCED COMBAT SYSTEM PROGRAM (SUBACS)

PMS 408 $14,400K (FY 82)

Figure 40

FFX DESIGN

PE 63564 **PROJECT S 1357**

FFX CONTRACT DESIGN PHASE - THE PRODUCT OF THIS EFFORT IS AN ENGINEERING DATA PACKAGE. MAY ALSO INCLUDE ENGINEERING DEVELOPMENT FOR COMBAT SYSTEM, SUBSYSTEM INTEGRATION, COMPUTER PROGRAM DEVELOPMENT AND TEST/EVALUATION SUPPORT.

SEA 03R **$4,041M(FY 82)**

Figure 41

ELECTRIC DRIVE

PROGRAM ELEMENT 63573N **SUB PROJECT S1314-SL**

DEVELOPMENT AND OPERATIONAL EVALUATION OF A FULL-SCALE 40,000 HORSEPOWER-PER-SHAFT ADVANCED ELECTRIC DRIVE SYSTEM FOR SHIPS. SYSTEM WILL CONSIST OF TWO 20,000-HP GENERATORS AND ONE 40,000-HP MOTOR PLUS ALL ANCILLARY SYSTEMS AND CONTROLS FOR OPERATIONAL EVALUATION AT A LAND-BASED TEST SITE. RECENT ADVANCES IN TECHNOLOGY PROGRAM FOR ADVANCED LIGHTWEIGHT, COMPACT AND EFFICIENT ELECTRIC MACHINERY PROVIDE THE OPPORTUNITY FOR MAJOR REDUCTIONS IN SHIP SIZE AND COSTS THRU THE ARRANGEMENT AND OPERATIONAL FLEXIBILITY OF ELECTRIC DRIVES.

COGNIZANT ACTIVITY	**FIRST YEAR (FY 82) FUNDING**
NAVAL SEA SYSTEMS COMMAND	$0.6M

Figure 42 (Note: 1 hp = 746 W).

ADVANCED ELECTRIC DRIVE

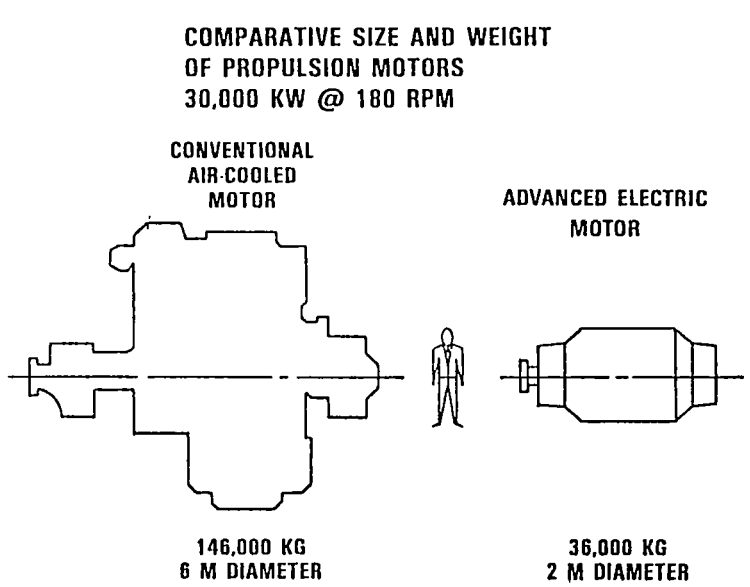


Figure 43

LIGHTWEIGHT SONAR

PROGRAM ELEMENT NO. 63589 **PROJECT NO. S1451**

AN/SQS-53 IMPROVEMENT PROGRAM PHASE II. THIS IS THE MODERNIZATION AND UPGRADE OF THE TRANSMIT AND RECEIVE SUBSYSTEMS. THE ESTIMATED R&D COST IS $130M. IOC IS PLANNED FOR 1989 CONCURRENT WITH DDG-X. THE TOTAL MARKET, INCLUDING BACKFIT, IS 120 SYSTEMS AT AN AVERAGE UNIT COST OF $13M (FY 80 $).

NAVAL SEA SYSTEMS COMMAND **FY 82 FUNDING $40.3M**

Figure 44

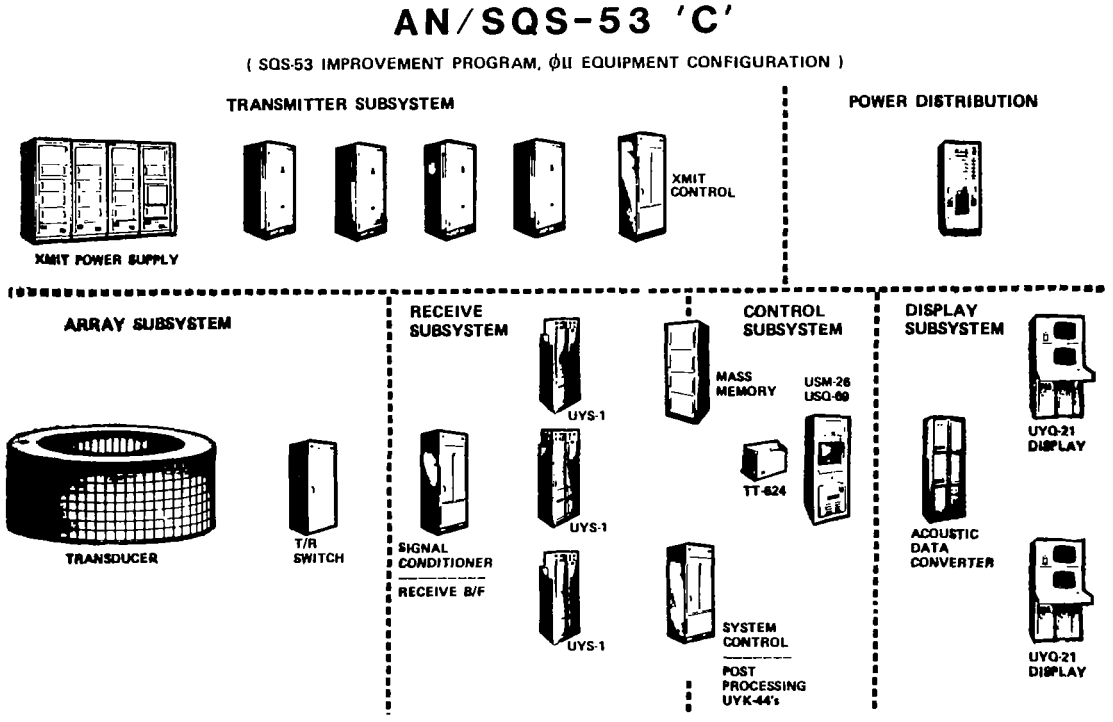


Figure 45

MOBILE ELECTRONIC WARFARE SUPPORT SYSTEM (MEWSS)

PE 63730 C0937

MEWSS WILL FULFILL THE REQUIREMENT TO PROVIDE AMPHIBIOUS ASSAULT AND RAPID DEPLOYMENT MECHANIZED FORCES THE CAPABILITY OF DETECTING, LOCATING, AND DEGRADING ENEMY TACTICAL AM AND FM RADIO COMMUNICATIONS IN THE HF, VHF, AND UHF FREQUENCY SPECTRUM. THIS CAPABILITY WILL BE INSTALLED IN AN AMPHIBIOUS ASSAULT VEHICLE, AND WILL BE COMPATIBLE WITH THE HIGH MOBILITY MULTIPURPOSE WHEELED VEHICLE (HMMWV).

NAVELEXSYSCOM 837K

Figure 46

ALL SOURCE IMAGERY PROCESSOR (ASIP)

63730 C1296

THE ASIP IS BEING DEVELOPED AS A REPLACEMENT FOR THE MAGIS AIR GROUND INTELLIGENCE SYSTEM (MAGIS) II SEGMENT AND WILL BE CAPABLE OF PROCESSING/ EXPLOITING MULTI-SOURCE IMAGERY AND SELECTED HARD COPY PRODUCTS.

DEVCTR MCDEC 788K

Figure 47

LIGHTWEIGHT BATTLEFIELD SURVEILLANCE DEVICE

P.E. 63730M C1421

THE LBSD IS AN X-BAND, MOVING TARGET INDICATOR, COHERENT DETECTION SURVEILLANCE RADAR SYSTEM. BASED ON A SERIES FERRITE FED ELECTRONIC SCAN ANTENNA, AN IMPATT TRANSMITTER, AND A PROGRAMMABLE SURFACE ACOUSTIC WAVE CORRELATOR/SIGNAL PROCESSOR, THE SYSTEM INHERENTLY POSSESSES MANY NEW CAPABILITIES FOR DAY/NIGHT, ALL-WEATHER TARGET ACQUISITION.

NOSC - San Diego NONE

Figure 48

LIGHTWEIGHT SEISMIC/ACOUSTIC PASSIVE BATTLEFIELD SURVEILLANCE DEVICE (LSAPD)

PE 63730M C1422

THE LSAPD IS A LIGHTWEIGHT, LOW COST, PASSIVE SURVEILLANCE SYSTEM REQUIRED TO DETECT AND LOCATE TARGETS OUT TO RANGES OF 10KM. THE DESIGN GOAL IS FOR A 34-KG, 2-10KM, MULTI-SENSOR, SEISMIC/ACOUSTIC, MAN TRANSPORTABLE FRONT LINE PASSIVE SURVEILLANCE CAPABILITY.

NAVAL OCEANS SYSTEMS CENTER
SAN DIEGO, CALIFORNIA

NONE
TRANSITIONS TO 6.3 IN FY84

Figure 49

OSD/OMB SUBMISSION

OCTOBER FYDP 18 SEP 80

6.4 PROGRAMS

PROGRAM TOTAL (BUDGET MINIMUM)	**$ 1,622.206M**
PROGRAM ELEMENTS	**98**
NEW P.E.s	**9**
PROJECTS	**284**
NEW PROJECTS (FY 82)	**28**

Figure 50

FY 82 NEW STARTS - PROJECTS (6.4)

P.E.	PROJECT	TITLE
64213N	W1502-SL	H-46 GPW SYSTEM
64219N	S1396-AS	ACOUSTIC PERFORMANCE PRED
64219N	W1442-AS	SH 2 RELIAB READINESS INSP
64226N	W1481-TW	ASPJ SUPT EQUIP
64226N	W1482-TW	ASPJ A/C INTEGRATION
64307N	S1275-AA	AEGIS PRODUCT IMP
64307N	S1447-AA	COMBAT SYS IMP
64314N	W0981-AA	AMRAAM
64352N	S0279-AA	MK 92 FCS UPGRADE
64353N	S1504-AA	VLS ASROC
64370N	S1500-AA	SSN-688 CLASS VLS
64505N	X1411-CC	SSN ICS
64514N	S1445-CC	DUAL MINI SINS IMP
64524N	S1347-AS	SUB ADV COMBAT SYS
64562N	S0366-AS	TORPEDO ENG DEV
64567N	S1357-SL	FFX
64646M	C1293-AW	ROTARY ENGINE

Figure 51

P.E.	PROJECT	TITLE
64657M	C1294-AW	FARS
64657M	C1443-AW	TRNG DEV/SIMULATORS (ENG)
64709N	Z1496-PN	TRI-SERVICE MNPR MGT
64715N	Z1426-PN	MOB ELECTR W/F SIM
64715N	Z1428-PN	AN/SQQ-23/BQR-20A OPTRNR
64725N	Z1433-PN	DYNAMIC SUB SYS SIM
64725N	Z1434-PN	SHIPBOARD C/S TEAM TRNR
64725N	Z1435-PN	SHIP HANDLING TRNR
64725N	Z1436-PN	SURF WARFARE TNG ANAL
64725N	Z1454-PN	DIGITAL RADAR TARGET SIM
64719N	C0053-CC	JTIDS

Figure 51.- Concluded.

VERTICAL LAUNCH ASROC

PE: 64353

SUB PROJ: S1504-AA

- MODIFY ASROC MISSILE FOR VERTICAL LAUNCH CAPABILITY
- MODIFY VLS TO INCORPORATE ASROC

COGNIZANT ACTIVITY

NAVSEA (PMS 410) – VLS

NAVSEA (63Y) – MISSILE SYSTEM

FY 82 FUNDING

$15.3 M

Figure 52

CURRENT VLS DEVELOPMENT PROGRAM

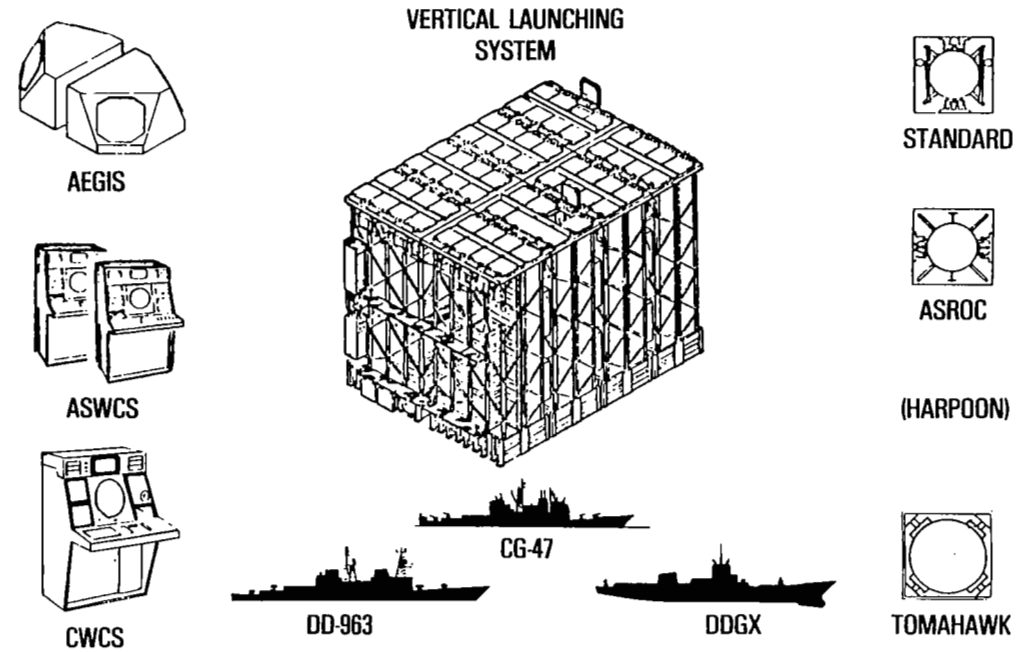


Figure 53

PROJECT TITLE: SUBMARINE ADVANCED COMBAT SYSTEM

PROGRAM ELEMENT NUMBER: 63524N, 64524N SUB PROJECT NUMBER: S-1346-AS, S-1347-AS

PROJECT DESCRIPTIONS:

- PROVIDE HIGHLY CAPABLE, INTEGRATED COMBAT SYSTEM FOR FUTURE SSN AND SSBN SUBMARINES
- COORDINATE ALL COMBAT SYSTEM AND SUBSYSTEM DEVELOPMENT EFFORTS TO OPTIMIZE SYSTEM LEVEL PERFORMANCE IN ALL WARFARE AREAS
- PROVIDE NEEDED OPERABILITY, RELIABILITY, AND MAINTAINABILITY IMPROVEMENTS
- REDUCED VOLUME AND LIFE CYCLE COSTS
- PROVIDE FOR EQUIPMENT AND SUBSYSTEM GROWTH POTENTIAL TO ACCOMMODATE TECHNOLOGY IMPROVEMENTS

COGNIZANT ACTIVITY:

NAVSEA PMS-409

FIRST YEAR (FY 82) FUNDING $M

	FY81	FY82
FY85 - SHIP START	6.3 12.0 (FYDP)	6.3 26.9 (FYDP)
		6.4 30.7 (FYDP)
FY86 - SHIP START	6.3 12.0 (FYDP)	6.3 26.9 (FYDP)
		6.4 30.7 (FYDP)

Figure 54

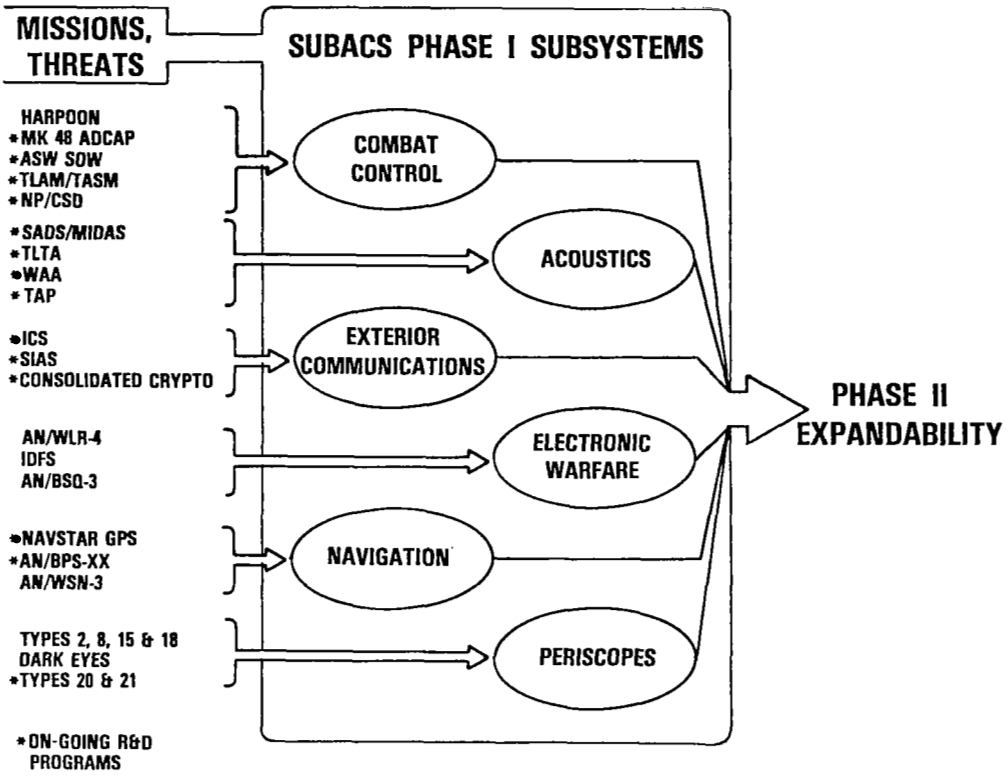


Figure 55

NAVAL TRAINING EQUIPMENT CENTER

P.E.	PROJECT	TITLE
63733	W1208	COMP GEN IMAGERY FOR SIM
63733	W1209	DYNAMIC SCENE VIS DISPLAY
63733	W1389	VTOL *VTRS LASER DISPLAY
63733	W1390	MULTI-SPECTRAL IMAGE SYS
63733	W1391	HELMET MOUNTED DISPLAY
64715	Z1426	MOB ELECTR W/F SIM
64715	Z1428	AN/SQQ-23/BQR-20A OPTRNR
64715	Z1433	DYNAMIC SUBSYS SIM
64715	Z1434	SHIPBOARD C/S TEAM TRNR
64715	Z1435	SHIP HAND TRNR
64715	Z1436	SURFACE WARFARE TRNG ANAL
64715	Z1454	DIGITAL RADAR TARGET SIM

*VTRS - VISUAL TECHNOLOGY RESEARCH SIMULATOR

Figure 56

COMPUTER GENERATED IMAGERY (CGI) FOR SIMULATION

PROGRAM ELEMENT: 63733N **PROJECT NO: W1208-PN**

DESCRIPTION

DEVELOP A CGI SYSTEM AND DATA BASE CAPABILITY UTILIZING ADVANCED TECHNOLOGY. PROVIDE ENHANCED SCENE DETAIL, VISUAL TEXTURE, AND MACHINE-INDEPENDENT DESCRIPTION OF THE VISUAL SCENE.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.505M

Figure 57

COMPUTER GENERATED IMAGERY FOR SIMULATION

TODAY'S TECHNOLOGY **TOMORROW'S GOALS**

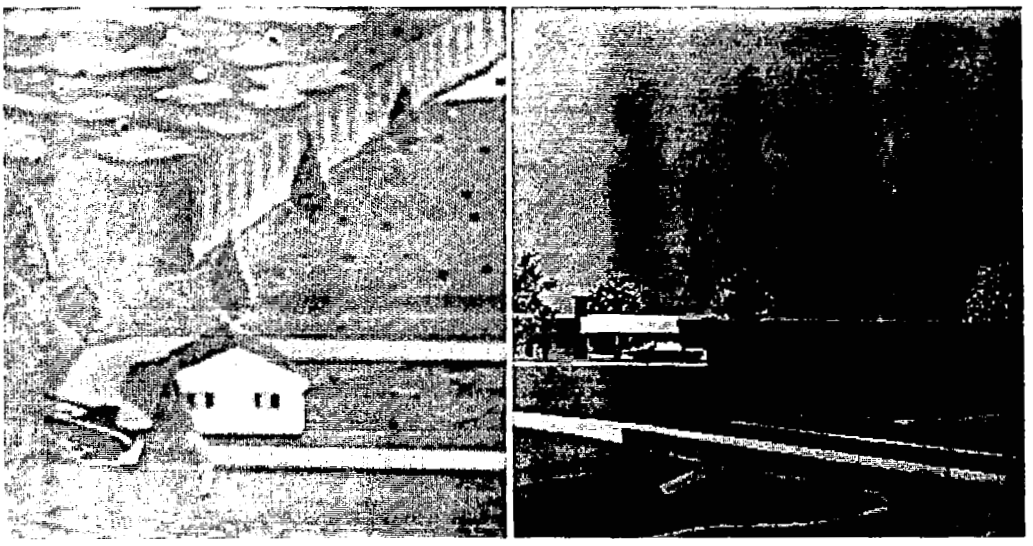


Figure 58

DYNAMIC SCENE VISUAL DISPLAY

PROGRAM ELEMENT: 63733N **PROJECT NO: W1209-PN**

DESCRIPTION

PROVIDE A DYNAMIC VISUAL DISPLAY THROUGH COMPUTER GENERATED IMAGERY OF SHIPS INTERACTING WITH WAVES, WEAPONS EFFECTS INCLUDING DAMAGE, AND OTHER EFFECTS SUCH AS SMOKE, MOVING SHADOWS, AND ILLUMINATION.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.504

Figure 59

APPLICATION OF DYNAMIC SCENE VISUAL DISPLAY TO SHIPHANDLING TRAINING

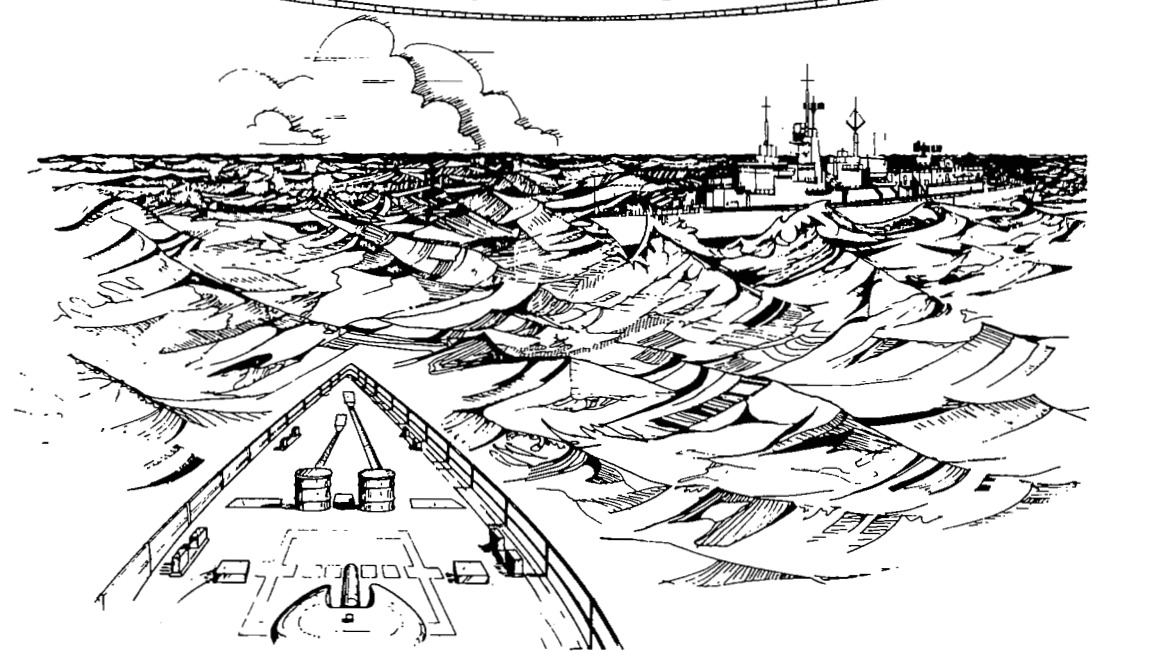


Figure 60

VERTICAL TAKE-OFF AND LANDING VISUAL TECHNOLOGY RESEARCH SIMULATOR LASER DISPLAY

PROGRAM ELEMENT: 63733N PROJECT NO: W1389-PN

DESCRIPTION

DEVELOP AN ADVANCED VISUAL DISPLAY SYSTEM FOR THE VTOL VTRS TO PROVIDE HIGH DETAIL LOW LEVEL VISUAL CUES CRITICAL TO VTOL OPERATIONS UTILIZING HIGH RESOLUTION SCANNING LASER DISPLAYS.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.403 (M)

Figure 61

APPLICATION OF VTOL VTRS LASER DISPLAY TO TERRAIN FLYING AND WEAPONS DELIVERY TRAINING

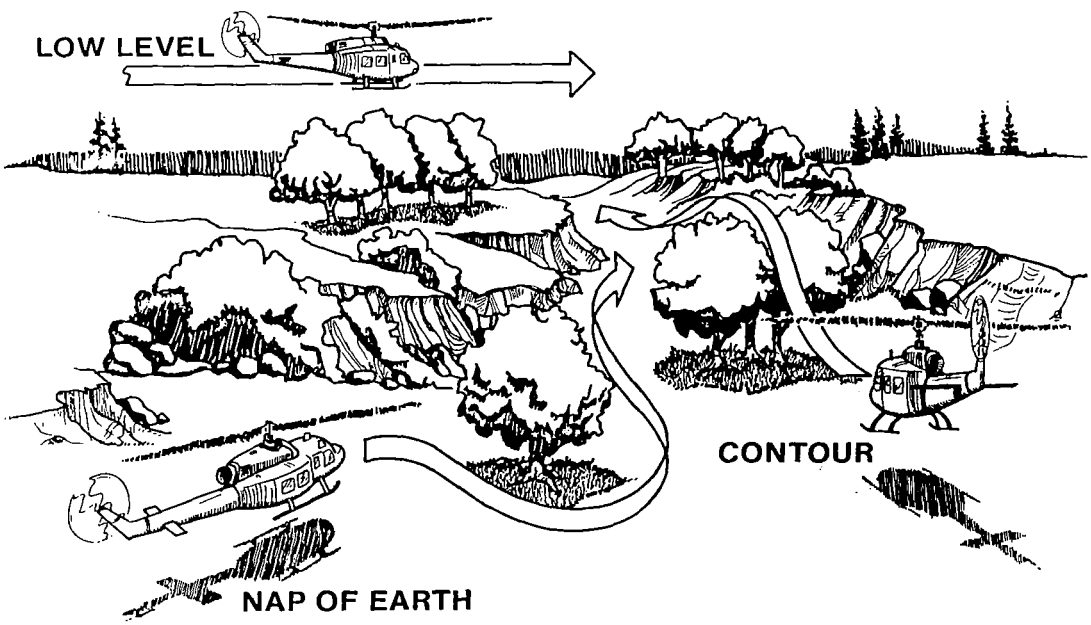


Figure 62

MULTI-SPECTRAL IMAGE (MSI) SIMULATION

PROGRAM ELEMENT: 63733N PROJECT NO: W1390-PN

DESCRIPTION

DEVELOP AN MSI SIMULATION FOR LOW LEVEL FLIGHT INCORPORATING FLIR, LLLTV, LASERS, AND RADAR. FEASIBILITY MODEL WILL DEMONSTRATE MISSION ORIENTED CORRELATION OF SENSORS.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.713 (M)

Figure 63

MULTI-SPECTRAL IMAGE SIMULATION

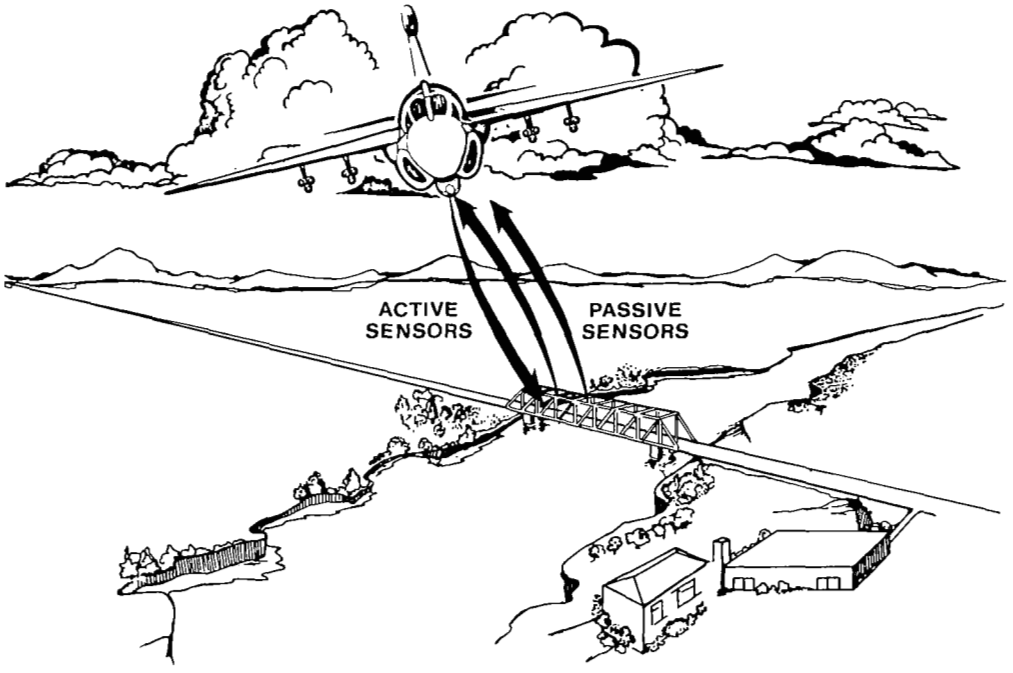


Figure 64

HELMET MOUNTED DISPLAY

PROGRAM ELEMENT: 63733N **PROJECT NO: W1391-PN**

DESCRIPTION

DEVELOP A WIDE FIELD OF VIEW, HIGH RESOLUTION VISUAL SIMULATION SYSTEM UTILIZING A HEAD/EYE AIMED PILOT HELMET MOUNTED LASER PROJECTOR.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $2.336(M)

Figure 65

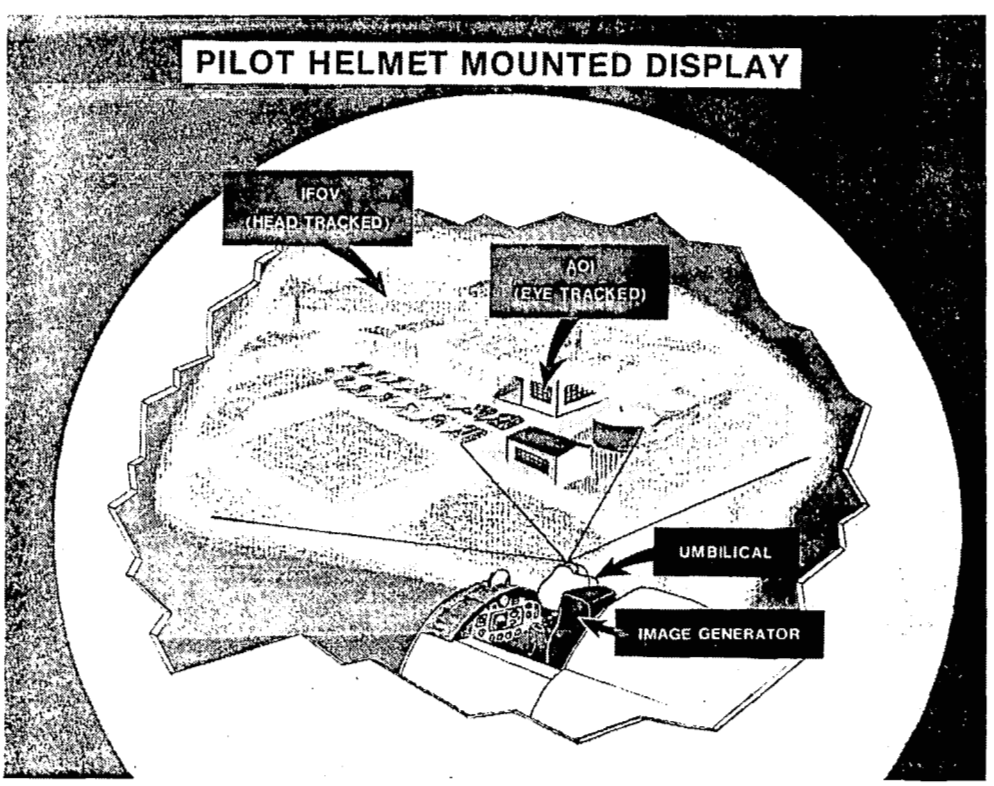


Figure 66

MOBILE ELECTRONIC WARFARE SIMULATOR

PROGRAM ELEMENT: 64715N **PROJECT NO: Z1426-PN**

DESCRIPTION

PROVIDES MANIPULATIVE SKILL TRAINING FOR ELECTRONIC WARFARE OPERATORS BY GENERATING AND TRANSMITTING ELECTROMAGNETIC SIGNALS FROM A MOBILE PLATFORM DIRECTLY TO THE ONBOARD SENSOR SYSTEM.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.8M
OPN: $2.3M

Figure 67

MOBILE ELECTRONIC WARFARE SIMULATOR

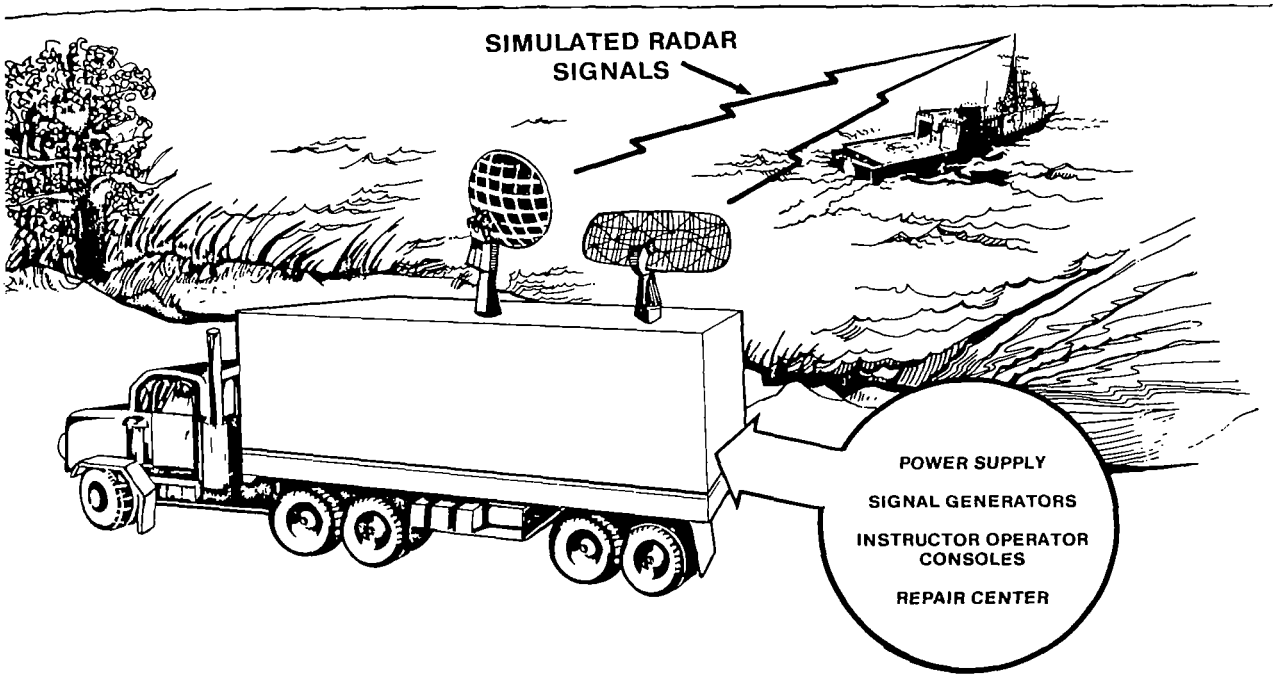


Figure 68

SQQ-23/BQR-20A
OPERATOR/TEAM TRAINER

PROGRAM ELEMENT: 64715N **PROJECT NO: Z1428-PN**

DESCRIPTION

PROVIDES AN OPERATOR/TEAM TRAINER THAT WILL ACCOMMODATE THE TRAINING OF PERSONNEL DESTINED TO OPERATE THE AN/SQQ-23 SONAR INTERFACED WITH THE AN/BQR-20A SONAR. THIS SYSTEM WILL MODIFY THE CURRENT DEVICE 14E24 AND ADD/INTERFACE THE BQR-20A TRAINING.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $1.9M

Figure 69

SQQ-23/BQR-20A
OPERATOR/TEAM TRAINER

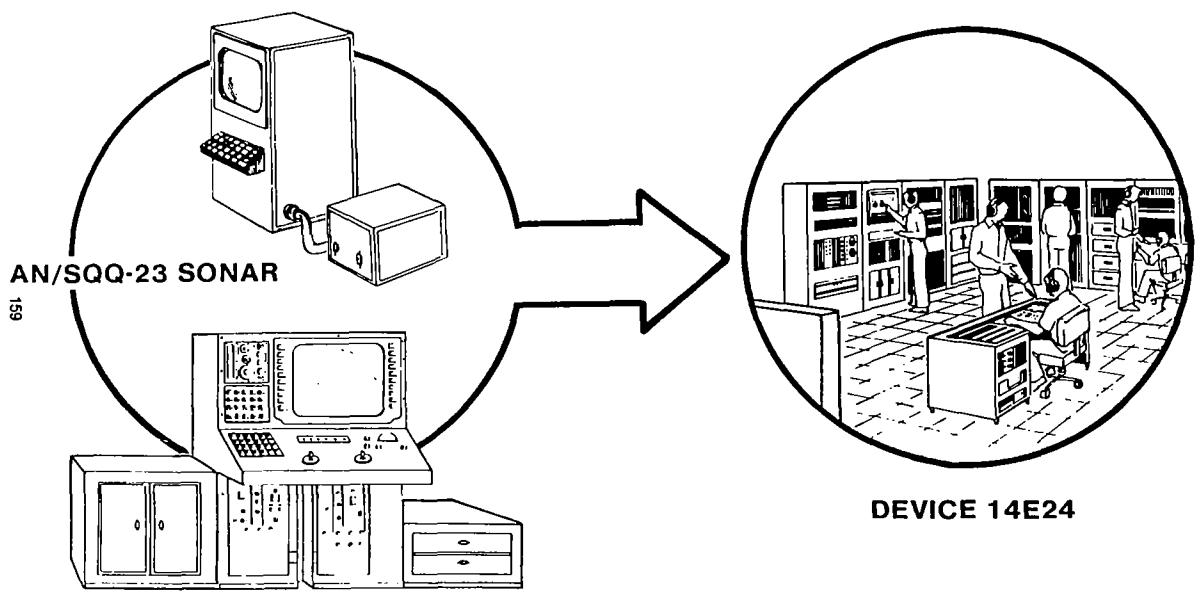


Figure 70

DYNAMIC SUBSYSTEM SIMULATION

PROGRAM ELEMENT: 64715N PROJECT NO: Z1433-PN

DESCRIPTION

PROVIDES FGG-7 COMBAT SYSTEM SOFTWARE SIMULATION FOR MAINTENANCE TRAINING TO BE INCORPORATED INTO THE CURRENT COMBAT SYSTEM MAINTENANCE TRAINING FACILITY AT MARE ISLAND.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $2.0M

Figure 71

DYNAMIC SUBSYSTEM SIMULATION

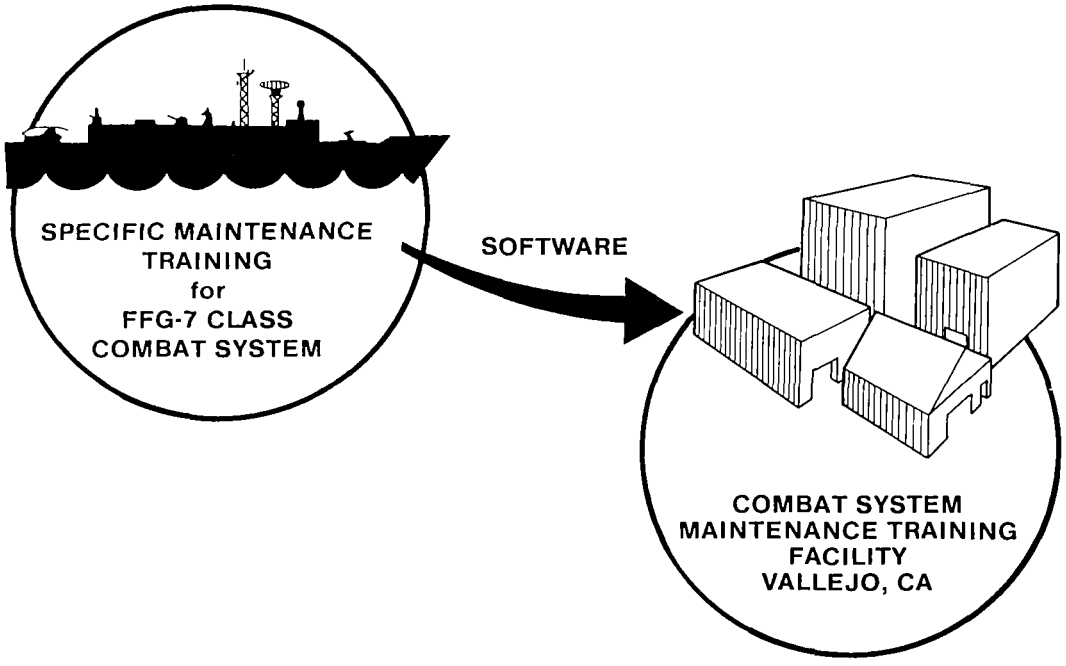


Figure 72

SHIPBOARD "ORGANIC" COMBAT SYSTEM TEAM TRAINER

PROGRAM ELEMENT: 64715N PROJECT NO: Z1434-PN

DESCRIPTION

DEVELOP EMBEDDED "ORGANIC" TRAINING SYSTEMS TO SUPPORT OPERATIONAL COMBAT SYSTEM TRAINING IN SURFACE COMBATANTS - THIS SYSTEM WILL PROVIDE INDEPENDENT COMBAT SYSTEM TRAINING AT SEA AND AT PIERSIDE.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.3M

Figure 73

"ORGANIC" COMBAT SYSTEM TEAM TRAINER

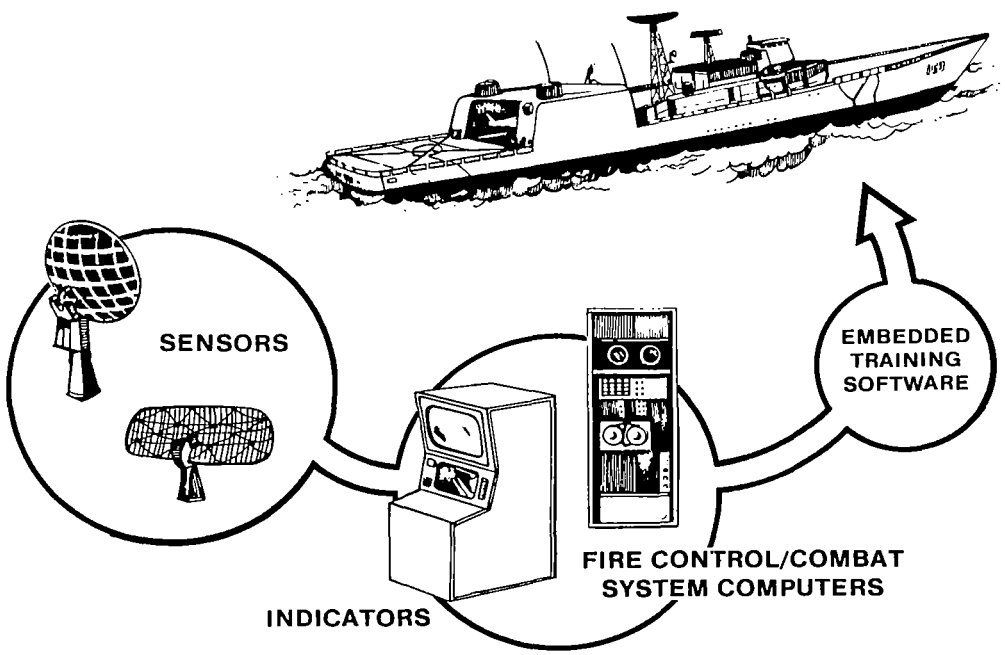


Figure 74

SHIPHANDLING TRAINING SYSTEM

PROGRAM ELEMENT: 64715N **PROJECT NO: Z1435-PN**

DESCRIPTION

DEVELOP A PART TASK AND ADVANCED SHIPHANDLING TRAINING SUITE TO PROVIDE REALISTIC TRAINING IN THE TWELVE FUNDAMENTAL KNOWLEDGE AND SKILL AREAS REQUIRED FOR PILOTING AND CONNING A SHIP.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.2M

Figure 75

SHIP HANDLING SIMULATOR

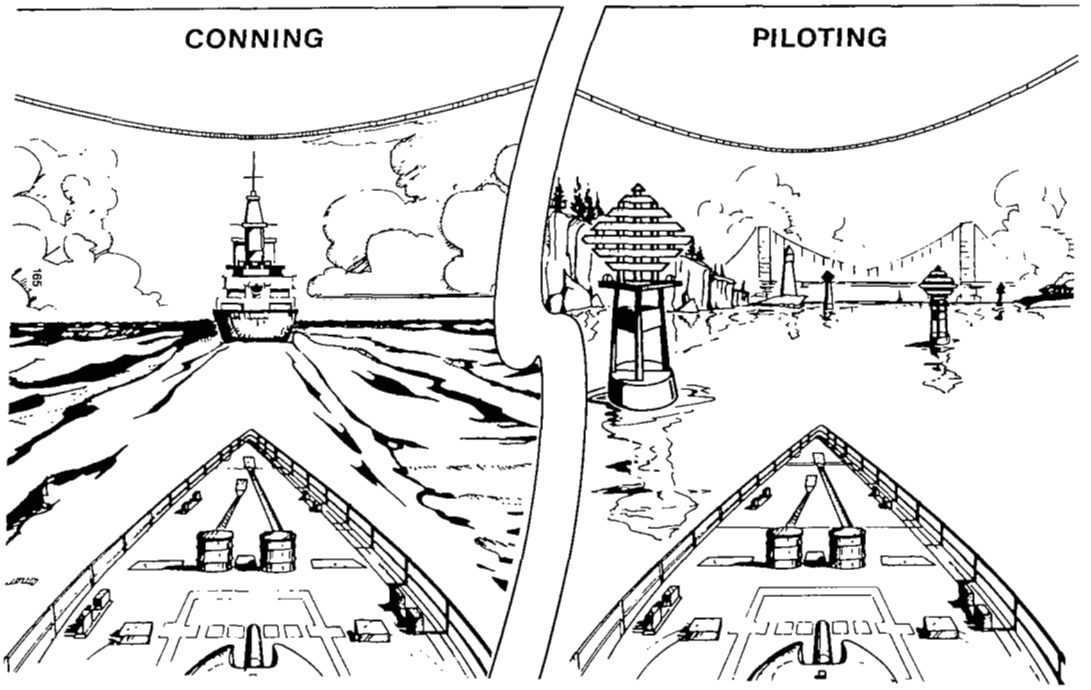


Figure 76

SURFACE WARFARE TRAINING ANALYSIS

PROGRAM ELEMENT: 64715N **PROJECT NO: Z1436-PN**

DESCRIPTION

PROVIDE FOR IN-DEPTH FRONT-END ANALYSIS OF SPECIFIC SURFACE WARFARE TRAINING PROBLEMS TO INCLUDE DEFININTION OF REQUIREMENTS/ SHORTFALLS, TRAINING OBJECTIVES AND STUDENT LOADING.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING: RDT & E: $0.2M

Figure 77

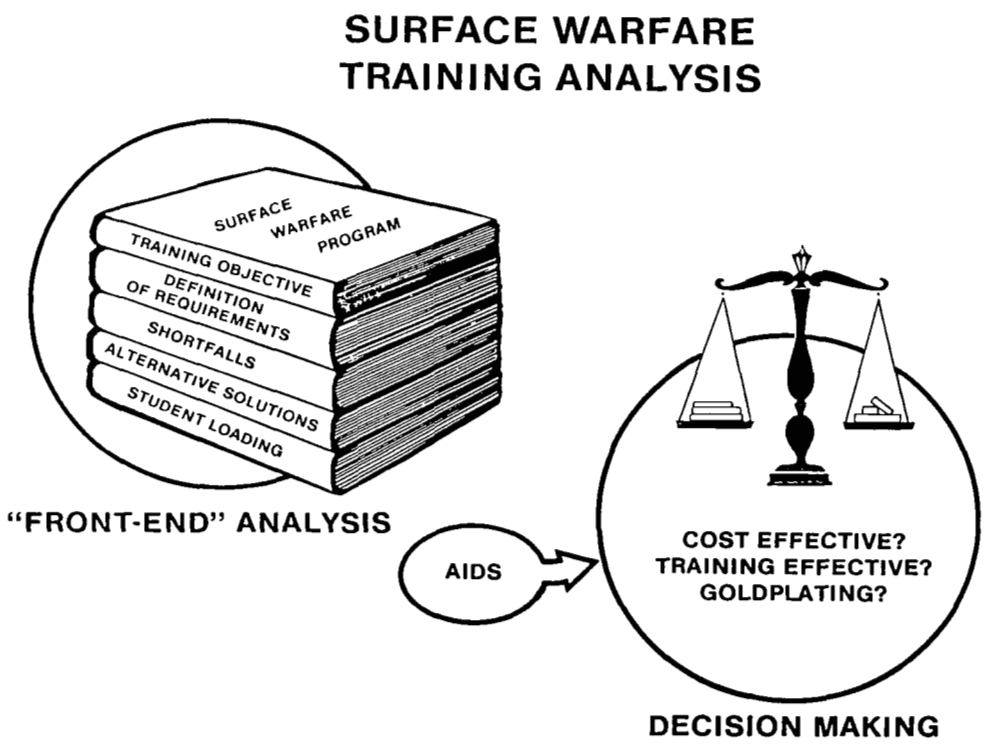


Figure 78

DIGITAL RADAR TARGET GENERATOR

PROJECT ELEMENT: 64715N PROJECT NO: Z1454-PN

DESCRIPTION

PROVIDES BASIC AIC/ASAC QUALIFICATION AND TRAINING TO PERMIT THE TRAINEE TO LEARN AND PRACTICE CONTROL OF VARIOUS SIMULATED OPERATIONAL AIRCRAFT. THIS DEVICE WILL SIMULATE RADAR AND IFF/SIF EQUIPMENT.

COGNIZANT ACTIVITY: NAVTRAEQUIPCEN

FY82 FUNDING:
RDT & E: $0.4M

Figure 79

DIGITAL RADAR TARGET SIMULATOR

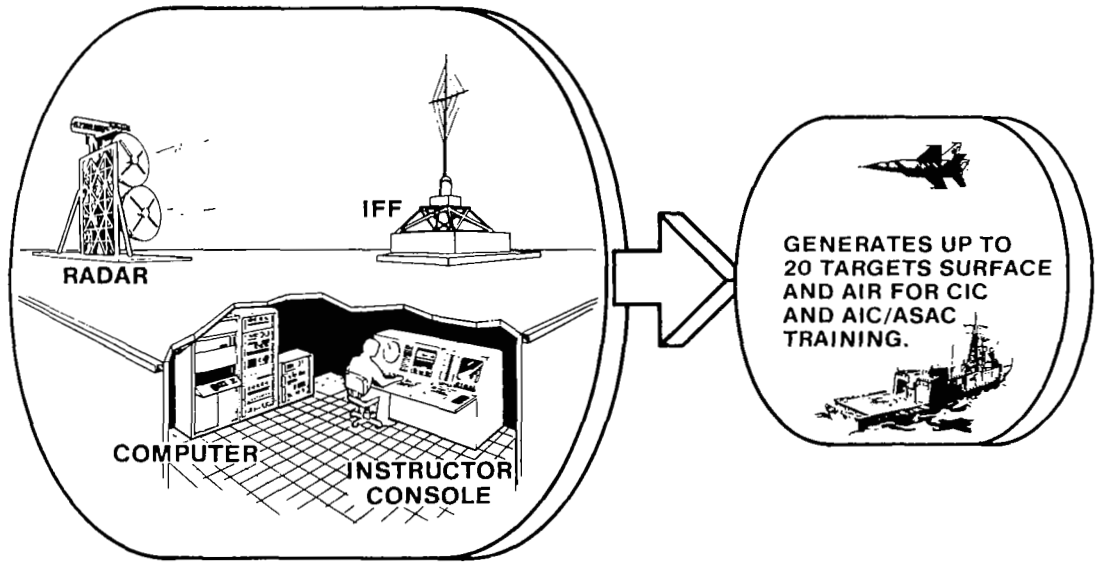


Figure 80

PERSONNEL RESEARCH AND DEVELOPMENT CENTER

P.E.	PROJECT	TITLE
63707	Z1383	CIV PERS ISSUES
63707	Z1385	COMPUTERIZED ADAPTIVE TEST
63710	Z1392	PERFORMANCE ENHANCEMENT
63720	Z1382	FUNCTIONAL CONTEXT TRNG
63720	Z1388	LOW COST MICRO COMP SYS
64709	Z1496	TRI SERVICE MNPWR MGMT

Figure 81

CIVILIAN PERSONNEL ISSUES

PROGRAM ELEMENT NUMBER: 63707N

PROJECT NUMBER: Z1383-PN

- DEVELOP NEW PERFORMANCE EVALUATION SYSTEM
  - IDENTIFY TASKS
  - MEASURE WORK OUTPUT
  - ESTABLISH PERFORMANCE STANDARDS
- CONDUCT SUPERVISORY TASK TRAINING

NPRDC

FY 82: $301K

Figure 82

COMPUTERIZED ADAPTIVE TESTING (CAT)

PROGRAM ELEMENT NUMBER: 63707N

PROJECT NUMBER: Z1385-PN

- COMPUTERIZED MILITARY SELECTION/CLASSIFICATION TESTING
  - REDUCED TESTING TIME
  - BETTER TEST ACCURACY
  - REDUCED CHANCE OF COMPROMISE/THEFT
  - AUTOMATED SCORING

NPRDC

FY 82: $301K

Figure 83

FUNCTIONAL CONTEXT TRAINING

PROGRAM ELEMENT NUMBER: 63720N

PROJECT NUMBER: Z1382-PN

- IDENTIFY JOB TASKS RELEVANT TO SELECTED RATINGS
- ALLOCATE JOB TASKS TO SPECIFIC COURSES
- STRUCTURE TRAINING TO PROVIDE:
  - INITIAL ORIENTATION
  - TRAINING OF JOB TASKS
  - WHOLE-TO-PART SEQUENCING OF TASKS

NPRDC

FY 82: $100K

Figure 84

PERFORMANCE ENHANCEMENT

PROGRAM ELEMENT NUMBER: 63710N

PROJECT NUMBER: Z1392-PN

- IMPROVE PERSONNEL PERFORMANCE IN SHIPBOARD SYSTEMS
- CURRENTLY EVALUATING SIMULATED ANTI-AIR DETECTION AND TRACKING TO:
  - IDENTIFY PROBLEMS
  - PROPOSE AND IMPLEMENT SOLUTIONS
- CONDUCT FOLLOW-ON R&D ON ADDITIONAL SHIPBOARD SYSTEMS

NPRDC **FY 82: $803K**

Figure 85

LOW-COST MICROCOMPUTER SYSTEMS FOR TRAINING

PROGRAM ELEMENT NUMBER: 63720N

PROJECT NUMBER: Z1388-PN

- EVALUATE PROTOTYPE MICROCOMPUTER SYSTEMS FOR TRAINING AS TO:
  - EFFICIENCY IN TEACHING DIVERSE SKILLS
  - COST EFFECTIVENESS

NPRDC **FY 82: $200K**

Figure 86

TRI-SERVICE
MANPOWER MANAGEMENT PROGRAM

PROGRAM ELEMENT NUMBER:
64709-PN

PROJECT NUMBER:
Z1496-PN

- **FUNDING FOR CRITICAL JOINT SERVICE RESEARCH IN:**
  - **MANPOWER**
  - **PERSONNEL**
  - **TRAINING**
- **MULTI-SERVICE PAYOFF REQUIREMENT**

OP-966D

FY 82: $4,800K

Figure 87

FY 82 NEW STARTS

NAVAIR

63207N	W1399	NOSS
63216N	W1401	HELO AIRCREW SURVEILLANCE
63217N	W0885	MOD AVIONICS PACKAGE
63217N	W0892	INFO HAND SYSTEMS
63262N	W0592	A/C & ORDNANCE SAFETY
63267N	W1253	NATO FUTURE IDENT SYSTEM
63308N	W0440	RAMJET MISSILE TECH
63313N	W0302	IR ATTACK WEAPON
63369N	W1446	MRASM (IIR)
63710N	W1230	DESIGN FOR MAINTAIN
63733N	W1208	COMP GEN IMAGERY FOR SIM
63733N	W1209	DYNAMIC SCENE VIS DISPLAY
63733N	W1389	VTOL VTRS LASER DISPLAY
63733N	W1390	MULTI-SPECTRAL IMAGE SYS
63733N	W1391	HELMET MOUNTED DISPLAY
63785N	W0646	ABN ELECTRO/OPTICAL C/M
63785N	W0659	E/O GUIDED WPNS C/M TEST
64213N	W1502	H-46 GPW SYSTEM
64219N	W1442	SH-2 RELIA READINESS INSP
64226N	W1481	ASPJ SUPT EQUIPMENT
64226N	W1482	ASPJ A/C INTEGRATION
64314N	W0981	AMRAAM

Figure 88

FY 82 NEW STARTS

NAVSEA

63506N	S0225	SURF SHIP TORPEDO DEF
63523N	S1332	SWATH
63524N	S1440	EMSP
63533N	S1417	SHPBD CORROSION CONTROL
63534N	S0308	SES
63536N	S0854	SOJS
63564N	S1357	FFX
63573N	S1314	ELECTRIC DRIVE
63589N	S1448	NON-AEGIS RADAR DEV
63589N	S1449	LIGHTWEIGHT AEGIS
63589N	S1450	COMBAT SYSTEM INTEGRATION
63589N	S1451	LIGHTWEIGHT SONAR
64219N	S1396	ACOUSTIC PERFORMANCE PRED
64307N	S1275	AEGIS PRODUCT IMPROVEMENT
64307N	S1447	COMBAT SYS IMPROVEMENT
64352N	S0279	MK 92 FCS UPGRADE
64353N	S1504	VLS ASROC
64370N	S1500	SSN CLASS VLS
64514N	S1445	DUAL MINI SINS IMP
64524N	S1347	SUB ADV COMBAT SYSTEM
64562N	S0366	TORPEDO ENG DEV
64567N	S1357	FFX

Figure 89

FY 82 NEW STARTS

NAVELEX

63520N	X1268	NAVY FUTURE COMM SYSTEM
63763N	X1319	TACT SURVEILLANCE SYSTEM
63784N	X0756	LTWT UNDERSEA SENS COMP
64505N	X1411	SSN ICS

NAVSUP

63710N	T1393	MICROFILM TECH FOR RECORDS

Figure 90

FY 82 NEW STARTS
MARINE CORPS

63635N	C1295	ARTY DIFIS
63730N	C0066	NON COMM ECM SYS
63730N	C0937	MOBILE EW SUPP SYS
63730N	C1296	ALL SOURCE IMAGE PROC
63730N	C1421	LTWT BATTLEFIELD SURV RADAR
63730N	C1422	LTWT SEIS/ACOUS PASS DEV
63731N	C0064	MAR INTEG PERS SYS
64646N	C1293	ROTARY ENGINE
64657N	C1294	FARS
64657N	C1443	TRNG DEV/SIMULATORS (ENG)

Figure 91

FY 82 NEW STARTS
CND

63707N	Z1383	CIV PERS ISSUES
63707N	Z1385	COMPUTERIZED ADAPTIVE TEST
63710N	Z1170	HUM PROC AUTO DATA BASE
63710N	Z1392	PERFORMANCE ENHANCEMENT
63720N	Z1382	FUNCTIONAL CONTEXT TRAINING
63720N	Z1388	LOW COST MICRO COMP SYSTEM
64709N	Z1496	TRI SERVICE MANPOWER MGMT
64715N	Z1426	MOB ELEC W/F SIM
64715N	Z1428	AN/SQQ-23/BQR-20A OPTRNR
64715N	Z1433	DYNAMIC SUB SYS SIM
64715N	Z1434	SHIPBOARD C/S TEAM TRNR
64715N	Z1435	SHIP HANDLING TRNR
64715N	Z1436	SURF WARFARE TRNG ANAL
64715N	Z1454	DIG RADAR TARGET SIM

Figure 92

FY 82 NEW STARTS

ONR

63371N	R1452	GEO SAT
63710N	R0126	OPERATIONAL DECISION AIDS
63785N	R0119	SURVEIL ENVIRON ACOUS SPT
63785N	R0120	TAC ASW ENVIRON ACOUS SPT

Figure 93

SOFTWARE ERROR DETECTION

Wolfgang Buechler and A. Gilliam Tucker
Comptek Research, Inc.
Santa Barbara, California

ABSTRACT

Effective software debugging requires capturing sufficient information when an error occurs to detect the primary source of error. This is particularly true with complex realtime systems where errors occur at unpredictable times and are difficult to recreate.

As part of the AN/SLQ-32 operational software, a large embedded real time system for the ROLM 1606, several methods were employed to detect both the occurrence and source of errors. The ROLM computer provides information about invalid memory addressing, improper use of privileged instructions, stack overflows, and unimplemented instructions. Additionally, software techniques were developed to detect invalid jumps, indices out of range, infinite loops, stack underflows, and field size errors. Finally, data is saved to provide information about the status of the system when an error is detected. This information includes I/O buffers, interrupt counts, stack contents, and recently passed locations.

These error detection techniques were a major factor in the success of finding the primary cause of error in 98% of over 500 system dumps.

INTRODUCTION

Effective software debugging requires capturing sufficient information when an error occurs to detect the source of error. This is particularly true with complex real time systems where errors occur at unpredictable times and are difficult to reproduce. Indications that an error has occurred at some previous moment are not adequate since determination of the original failure may often be impossible. To be effective, software error detection logic must detect and expose the primary error condition as soon as possible to maximize the availability of useful diagnostic information.

As part of the AN/SLQ-32 operational software, a large embedded real time system for the ROLM 1606, several methods are employed to detect both the occurrence and source of errors. These techniques include error-related events and information made available by the ROLM computer itself as well as error detection and diagnostic logic installed in the software package.

The AN/SLQ-32 real time electronic warfare command and control system is capable of detecting multiple radar signals at very high data rates. The central computer is a ROLM 1606 with memory configurations ranging from 64K to 112K words. Access to the memory in excess of 64K is accomplished through dynamic memory map switching evoked by a special purpose executive module. The entire operational software is written in assembly language and structured into a multi-task/multi-user configuration. Inter-task communication and coordination are accomplished via system calls to the executive module. Integral to this executive module structure is the implementation of error detection techniques to exploit the ROLM error processing and effect additional software processing. Several errors are recognized and corrective action is attempted on-line. However, the majority of errors are considered fatal and the system halts in an orderly error shutdown mechanism. The various errors detected include:

- unimplemented instructions
- stack overflows
- memory address violations
- jumps to an invalid location
- stack underflows
- task using excessive time
- array index out of bounds
- data wrong size for field
- invalid executive request
- unscheduled return
- unknown interrupts
- privileged instructions

UNIMPLEMENTED INSTRUCTIONS

The unimplemented instruction trap occurs when the ROLM 1606 CPU encounters a bit pattern which does not decode into a recognizable instruction. This trap accounts for less than one percent of all system halts. It is typically caused by executing data rather than program, memory failure, or power supply instability. If initialized properly the ROLM 1606 automatically traps to error routines provided by the operating system.

STACK OVERFLOWS

A stack overflow trap occurs when more data is pushed onto a ROLM 1606 supported stack than the stack can hold. This error accounts for less than one percent of all system halts. It is typically caused by a subroutine's calling itself, re-entrance problems, or interrupt masking problems. The stack overflow trap is similar to the unimplemented instruction mechanism.

MEMORY ADDRESS VIOLATIONS

When a software program attempts to access a memory location which is designated invalid to access in the ROLM 1606 map descriptor table a memory address violation occurs. This trap accounts for approximately 19 percent of all system halts. Typical causes of this violation are undefined variables, improper indexing of stack items, or bad table indices.

Diagnosis of these problems is made easier by the ROLM 1606 last address file (LAF) which contains the following four values:

1) Location of last instruction
2) Location of next-to-last instruction
3) Address of last data fetch/store
4) Address of last DMA fetch/store

At the time of the memory address violation the LAF is frozen and the addressing information is available to the executive error processing. The original software error can then be deduced from the program information which is generally intact.

JUMP TO AN INVALID LOCATION

A jump to an invalid location occurs when the CPU is directed to change the program counter to a location not within the normal program flow. Particularly common and also difficult to diagnose is a jump to location zero. Failures of this class account for approximately 9 percent of the system halts. Typical software errors causing this type of failure include instruction overwrite, improper index of stack, or improper handling of return linkages. The AN/SLQ-32 operational software was modified to include an instruction at location

zero and several unused locations to cause a memory address violation when executed. This freezes the last address file before the next-to-last instruction fetch is modified. This address indicates the offending jump quickly and reliably.

STACK UNDERFLOWS

A stack underflow is the attempt to pop more data off a stack than is currently contained on that stack. Sometimes incorrect data is read while at other times a random variable would be used as a program counter as with RTRN or PRT instructions. Less than 1 percent of the systems halts are stack underflows. Typical underlying software errors include poor stack access, jumping into the middle of routines, or incorrect return from routines. To detect the incorrect use of a PC due to a stack underflow as soon as possible, the area after the stack is padded with the address of an executive stack underflow detection routine. When this address is popped off as the PC the error routine takes control and allows the programmer to examine the software state at the time of the error.

TASK USING EXCESSIVE TIME

The AN/SLQ-32 operational software is a non-pre-emptive task structure. Specifically, the executive allows each task to retain control of the CPU (excepting interrupts) until completion of the task's function. Nominally, the maximum design task time is approximately 100-200 msec. If the task retains control for a significantly longer time than the nominal, the executive declares that the task is using excessive time. This failure type accounts for approximately 15% of all halts. Typical causes of this failure are infinite loops, searching infinite linked lists, and constantly interrupting hardware. The executive monitors for this error by keeping a time within task counter driven by the one msec real-time clock interrupt. Capturing this event does not guarantee the retention of more software data specifically, but it does allow for an orderly, recognizable software shutdown.

ARRAY INDEX OUT OF BOUNDS

In any file structure an accessing program may attempt to fetch data from beyond the end of a table. Typically, this may be caused by tables which are too small, unexpected external conditions, or improper passing of table indices. These failures account for a surprising 36% of all SLQ-32 system halts. Without earlier detection the original reason for the failure would be greatly masked and nearly impossible to diagnose. The SLQ-32 software has been augmented to use common data handlers for most array references. These handlers which are generated by macro code incorporate limit checking code for the array index and trap to the executive error handling for notification to the programmer.

DATA WRONG SIZE FOR FIELD

When a software program stores values into a data base there is a chance that the field size allocated to the data field may be smaller or larger than the data. This error may be caused by bad links in linked lists, programs which do not

check for bad values, and generated data values which exceed design expectations. Such errors account for approximately 15% of the system halts. As with array indices out of bounds it is essential that the error be trapped immediately before significant changes are made to the error environment. The SLQ-32 common data handlers incorporate a check of field size versus data size and trap to the executive on violations.

INVALID EXECUTIVE REQUESTS AND UNSCHEDULED RETURNS

As the executive module performs services for the software tasks, it requires input of various types. This input may be erroneous. Errors of this type account for approximately 2 percent of all system halts and are typically caused by out-of-sequence operations or non-re-entrant interrupt handler code. By checking the validity of all input the executive traps bad service requests.

UNKNOWN INTERRUPTS

When a device code is presented to the ROLM 1606 auto branching interrupt sequence a branch will be made to the appropriate interrupt handler. If this device code is not normally expected the event may be considered as an unknown interrupt. Typically, unknown interrupts are due to hardware failures. If the interrupt is merely a single time occurrence which does not tie up the interrupt request line, it can be reported by the executive and ignored. However, if more than a specific number of interrupts occur within a certain unit of time then the situation is considered to be a high frequency unknown interrupt and processing is halted. This allows the tactical operator a notification of the problem and he may attempt to restart or choose to run diagnostics.

PRIVILEGED INSTRUCTIONS

The SLQ-32 software structure allows each task the capability to execute such privileged instructions as INTEN or INTDS implemented through device code 77. Additionally, the display data to panel instructions is trapped by the executive and executed for the users. All other privileged instructions are not allowed in user mode. Errors of this type occur for reasons similar to the unimplemented instruction trap.

STATUS INFORMATION

In addition to stopping program execution as soon as an eventually fatal error is detected the SLQ-32 operational software attempts to save as much historical data as efficiently possible. Register contents are saved at a halt along with a derived PC and other status indicators. The software keeps track of the last 100 key locations encountered, the number of interrupts per device code, the count and location of DMA violations, and the last message sent to each task. Additionally, for program performance analysis, the number of entrances per task, maximum time per task, and average time per task are maintained. This data often gives good clues as to problem solutions.

ROLM 1606 SIMULATOR

All the techniques so far discussed assist in debugging problems in a full system configuration. Ideally most problems should be found in new or modified code prior to actual insertion into an existing system load. To accomplish adequate unit testing of individual segments of software code (i.e., routine or set of routines) when a ROLM 1606 computer is not available or access is severely limited, a 1606 simulator has been developed to run under the AOS operating system on a Data General ROLM Eclipse computer. This simulator interacts with a user at a standard AOS terminal. The save file image is executed from a disk file and utilizes AOS paging algorithms to execute within one 1 KW pages.

A large range of debugging commands is available including:

- breakpoint
- instruction step
- start/continue
- panel instruction
- deposit/examine memory locations
- deposit/examine status indicators
- deposit/examine accumulators and registers
- initiate/print jump trace
- timing trace
- halt

Use of the 1606 simulator to unit test software segments allows controlled testing of all decision paths. Such common errors as badly encoded tests and improper access of data via index or indirection are easily found. Complete control of the software environment encourages more exhaustive testing in various data configurations. Timing problems are generally difficult to uncover with this method, but clever testing can be used to force certain re-entrance situations. Since the 1606 simulator runs under AOS, several users may debug independently at the same time and so no longer be dependent on actual ROLM 1606 computer availability schedules for initial testing.

CONCLUSION

Effective problem resolution requires that errors be detected as close to the primary source of error as possible. Whenever tactically possible, execution should be halted and a core dump taken or equivalent data extracted. These requirements are confirmed by the successful solution of 98% of 500 separate SLQ-32 system halts. Ideally, software errors should be solved before insertion into a working system. Segment simulation on a non-target computer facilitates this goal.

ARTS BETA TESTING REPORT

Michael C. McCune
Command, Control, and Communications Corporation
Torrance, California

ABSTRACT

Command, Control and Communications Corporation (4C) has been a test site for the ROLM Advanced Real Time System (ARTS). Our tests utilized existing commercial system hardware and software, which has been operating under AOS for several years in a multitasking, multiprocessing, and multiple computer environment. This paper will discuss our experiences with ARTS in terms of compatibility with AOS, ease of transmission between AOS and ARTS, and functional areas of ARTS which were tested. Relative and absolute performance of ARTS versus AOS as measured in our system environment will also be presented.

1. INTRODUCTION

Command, Control and Communications Corporation (4C) has conducted BETA testing of the ROLM Corporation Advanced Real Time System (ARTS). This paper discusses the goals, methodologies, and results of this BETA test project. In the paper, there is an introduction to 4C, a definition of the characteristics of the real time systems which 4C develops and produces, an establishment of the need for such an operating system as ARTS, and a report on the conduct and results of the ARTS BETA testing.

2. INTRODUCTION TO 4C

4C was established in 1972 as a systems house for tactical systems and applications. 4C's primary products are special purpose computer systems for simulation (data link, radar, IFF), tactical applications (such as data link buffers, radar processing systems, tracking systems), and test beds (for JINTACCS testing, operational effectiveness evaluation). 4C also performs software development and hardware development and has produced a number of products, such as a CMS-2M compiler for ROLM and Data General computers, networking software for these computers, and numerous software tools. Hardware products, besides complete systems, include tactical data link modems and buffers, radar and IFF target extractors, and numerous other computer interface devices. 4C customers include the U. S. Services, U. S. Joint Service projects, NATO countries, foreign military sales (FMS), and other countries.

3. NEED FOR A REAL TIME AOS

Over a period of time, the requirements for the various 4C system products have evolved to become quite demanding and complex. As shown in Appendix A, (Figure 1), the requirements were initially satisfied with Data General NOVA and ROLM 1603 processors. These

early systems had a single computer and a single process to execute on that computer.

All the programs were memory resident; the programs were written in assembly language; the system performed a single, fixed function, or a single set of fixed functions; and the arithmetic requirements were generally satisfied by fixed-point arithmetic. Over the years, the requirements have grown to the point that the software is now multiple processes which execute on one or more computers; these processes are primarily memory resident, but may have some programs which operate on demand; the programs are now written in several languages, including 4C's implementation of CMS-2M, FORTRAN, and Assembly language; the system performs a number of fixed functions and may include additional (variable) functions as an outgrowth of specific customer requirements; and, finally, the computational environment now requires a mix of fixed point and floating point arithmetic. Figure 2 (Appendix A) depicts the configuration of a typical large system. In this system there are several processors: one or more ECLIPSE processors are present as the major data processors in the system, and there are one or more NOVA-type processors that are used for controlling the unique input/output devices required in the system configuration. All processors are interconnected via a Multi-processor Communications Adaptor (MCA).

In the typical system configuration, the NOVA computers run a proprietary executive program developed by 4C, and the ECLIPSE computers are under the control of the Data General Advanced Operating System (AOS). This operating system was selected because the computing environment for our systems was more demanding than could be satisfied by RDOS and because the Advanced Operating System offered significant improvements in flexibility and capability over RDOS. Figure 3 (Appendix A) depicts the typical hardware environment for an AOS system. Of course there is an ECLIPSE CPU and its memory. The most notable requirement

from a MIL-SPEC system or multi-processor system point of view is that AOS requires not only a terminal, a real time clock, and a program interval timer, but it also requires a disk and a mag tape or diskette. The single processor environment might be expected to have all of these devices, but in a multi-processor environment or a MIL-SPEC environment the requirement for disk and mag tape is unfortunate, because these devices are not only expensive but are comparatively unreliable, large, and heavy.

AOS is also a commercial operating system which has been designed to support a wide variety of system capabilities and modes of operation. Because of this flexibility and generality, users of AOS pay a certain penalty in terms of system throughput and overhead. In a real time application, the system overhead of AOS is, at the minimum, undesirable and may, in fact, be unacceptable.

The need for a real time AOS is, therefore, based upon the following requirements. First, we want higher system throughput than is possible with AOS. This increase in throughput must be achieved without changing application software, without changing utility software (such as 4C's CMS-2M compiler or networking software), and without requiring extensive retraining of programmers and support personnel. Second, we want to eliminate the need for the disk and mag tape unit for every computer in a system configuration. While the system requirements may require one or more of these devices for the system mission, it is very undesirable that the operating system itself require the presence of these devices on every ECLIPSE or MIL-SPEC ECLIPSE CPU. Third, we want a smaller and more configurable AOS so that the memory consumption of the operating system is both minimized and optimized for the application. Figure 4 of Appendix A summarizes a number of characteristics that would exist for a real time AOS.

4. ARTS BETA TEST PLAN

In the first half of 1980, 4C began discussions with ROLM Corporation about the ARTS system, which was then under development. 4C had firmly established its internal need for a product such as ARTS, but a survey of available operating systems proved that there was no satisfactory replacement as yet available for AOS. ROLM, on the other hand, had developed the MIL-SPEC ECLIPSE and had also perceived the need for a real time version of AOS. Since the ARTS development so closely matched 4C's need for an advanced real time operating system, and because 4C was uniquely experienced in real time applications with AOS, 4C and ROLM agreed to establish a BETA test project for ARTS at 4C headquarters in Torrance, California.

The ARTS BETA test project at 4C was established with four initial goals. First, we wanted to verify the compatibility of ARTS and AOS in the context of 4C's system applications. The second goal of the BETA test project was to identify all problems encountered during the testing of ARTS and forward the problem description to ROLM for correction. The third goal was to verify that problems had been corrected when new updates of ARTS were returned from ROLM. The fourth goal, and perhaps the most important one, was to determine the relative and absolute performance of ARTS and AOS in terms of a number of important characteristics, such as system overhead, scheduling delays, and the execution times of common system calls.

A five-step approach was established for conducting the BETA test project. The initial step was to install ARTS and become sufficiently familiar with its use so as to be self-sufficient. The second step was to perform some basic operational checks on ARTS to verify that it was ready to be used in more detailed testing. The third step was to perform a set of detailed compatibility and integrity tests. This step used an existing set of test programs

which performed an involved set of inter-task, inter-process, and inter-CPU data exchanges. The nature of this test was such that it exercised a majority of the system features which are relied upon in the 4C systems. The fourth step was to perform "side-by-side" functional timing tests for AOS and ARTS. These tests would be accomplished using programs developed specifically for this purpose. The functional timing tests would not only verify that ARTS and AOS can execute the same programs, but the test program outputs would provide relative and absolute measures of the performance, throughput, and/or timing of typical AOS and ARTS functions. The fifth and last step of the data test project would be to perform a "side-by-side" system performance analysis of an actual 4C real time system. This test would use measuring techniques such as histograms and idle-time measurements to determine the relative responsiveness and throughput of an actual system operating under ARTS and AOS.

5. BETA TEST RESULTS

The BETA test project was initiated over a period of a week. During this week there was a team of four people from ROLM on site at 4C headquarters in Torrance. This team performed the initial installation of ARTS and provided a short introduction to using ARTS and configuring ARTS for specific system applications. During the subsequent days, an intensive effort was made by the ARTS installation team and 4C representatives to create and verify an initial operational capability for ARTS. This effort was successfully completed within that time, and an existing 4C system which had been running under AOS was brought up and demonstrated operating under ARTS (see Figure 5., Appendix A). This initial system was a 4C SIMTRACC configuration which included a graphic display, the operation of tactical data links (TADIL-B and ATDL-1), and online data collection and data reduction operations. The system involved five interrelated user processes and had a number of "IDEF" devices within the system. It was agreed by ROLM

and 4C that this initial installation and operation was highly successful. Although a number of start-up problems were discovered, the team operated well and the majority of problems were corrected on the spot.

After the initial installation efforts, 4C entered into a compatibility and integrity test phase. This test phase consisted of exercising a variety of ARTS system capabilities using some existing functional test which had been previously developed by 4C for testing its proprietary network transaction package. This test was considered to be a reasonably exhaustive exercise of ARTS capabilities (from 4C's point of view) in that it included the following elements: inter-task communications, inter-process communications (using IPCs and shared files/pages), inter-CPU communications (using MCAs), and file input and output. Results of these tests have been summarized in Figure 6. During this test phase, the inter-task communications worked immediately. The inter-process communications, inter-CPU communication, and file I/O portions of the test each found some errors within ARTS. As these errors were discovered, they were identified to ROLM, and ROLM responded with a combination of patches and subsequent releases so that in a reasonable period of time all errors were corrected and all tests were fully operational.

At this point, it was deemed worthwhile to begin the functional timing tests comparing ARTS and AOS. It was felt that ARTS had achieved sufficient maturity so that the ARTS configuration which we had in house (pre-release version Ø.Ø5) would provide meaningful timing results. Special timing tests were devised for the following system functions: (a) system overhead; (b) system scheduling overhead; (c) inter-process communications (IPC) throughput; (d) ?XMT throughput; (e) ?REC throughput; (f) ?XMT/?REC throughput; (g) character I/O output rate; (h) file use (OPEN/READ/CLOSE) throughput; (i) block I/O (?RDB/?WRB) throughput; (j) shared page re-map throughput. These tests were all designed so that they would provide useful results regardless of

the processor configuration details, such as memory size, memory interleaving, processor type, or processor peripheral configuration. The implementation of each of these timing tests is described below.

(a) System Overhead Test. This test operates as a single process with two tasks. The first task controls the initialization of the second task and also provides the basic timing interval for the measurement. The second task consists of an idle loop of known and fixed construction. The purpose of the test is to count the number of times that the idle loop can execute in a known period of time. The amount of available time in the system can then be expressed in terms of the number of loops per second. In addition, an identical timing loop was established in a stand-alone system, and the computer hardware capacity for executing the timing loop was established. These two figures were sufficient to determine the percentage overhead of the operating system and from that to determine the amount of CPU time which is available for application processing. The system overhead and available time figures thus derived were used in other tests in determining the time required to execute other system functions. Figure 7 of Appendix A summarizes the test results for the System Overhead Test.

(b) System Scheduling Overhead Test. This test consisted of using the system overhead test described above, operating in the presence of 1, 2, and 3 other processes. By determining the amount of available CPU time when 1, 2, and 3 other processes are executed, it is possible to determine the amount of time spent scheduling the operation of the background processes. The background processes all executed the same code which was a single

task. This task performed a ?DELAY system call specifying an activation rate of one activation per clock tick. Therefore, each of these processes represented a known frequency of operation. In addition, this test was varied across three standard clock rates (10, 100, and 1000 Hertz) so system overhead itself was known at the common clock rates. Figure 8 of Appendix A summarizes the test results for the System Scheduling Overhead Test.

(c) IPC Throughput Test. This test executed as two processes. One process performed an echo function wherein it received an IPC message and immediately returned it to the sender. The second process operated as two tasks; one task performed the measurement interval control and the second task initiated the IPC message to the echo task and receiving the response from the echo task. The output of this test is a measurement of the number of cycles of IPC send/receive which can be performed in one second. Figure 9 of Appendix A summarizes the results of the IPC Throughput Test.

(d) ?XMT Throughput Test. This test and test (e), which is the ?REC Throughput Test, were used to establish some baseline information in support of test (f). The ?XMT test and ?REC test were a one-process, two task test which measured the number of times that a ?XMT and ?REC system call could be performed. Because there is a decision function implicit in the ?XMT and ?REC processing, the mailbox was kept permanently empty for the ?XMT test and permanently full for the ?REC test. Figure 9 of Appendix A summarizes the results of the ?XMT and ?REC Throughput Tests.

(e) See (d).

(f) ?XMT/?REC Throughput Test. This test was performed as a single process with three tasks. One task performed initialization and test measurement interval control, while the other two tasks operated as a ping-pong, multi-tasking test. Task A would do a ?XMT to awaken Task B and then a ?REC. Task B would then do a ?XMT operation. The output of this test was a count of the number of complete cycles which were executed in one second. Figure 9 of Appendix A summarizes the results of the ?XMT/?REC Throughput Test.

(g) Character I/O Output Rate Test. This test was intended as a measure of system overhead and as a measure of responsiveness to interrupts when performing interrupt-character output. The program operated as a single process with two tasks. Task 1 performed initialization and test measurement interval control, while Task 2 operated in a hard loop outputting fixed length records of ASCII characters to the system control console, known as @CONØ. The results of this test would be the number of characters per second which could be output on a 9,6ØØ-bit-per-second, serial asynchronous line. A maximum output rate would be 96Ø characters per second. The results of the Character I/O Output Test are found in Figure 10 of Appendix A.

(h) File Use Throughput Test. This test is intended to indicate the relative performance of the operating system in performing an OPEN/READ/CLOSE cycle. A file is opened, 5Ø records of 8Ø bytes each are read, and then the file is closed. The output of this test consists of a count of cycles which can be executed per minute. Figure 10 of Appendix A contains the results of the File Use Throughput Test.

(i) Block I/O Throughput Test. This test measures the number of ?RDB and ?WRB system calls which can be executed per second. For the purposes of this test, the block I/O device was selected to be an MCA. This device was selected because, with the cooperation of a second CPU, it is possible to use the MCA as a zero-latency DMA block I/O device. The output of this test is the number of ?RDB and ?WRB calls which can be executed per second. Now this figure is of importance because the ?WRB and ?RDB calls are the most basic element in performing I/O to DMA devices, such as disks, tapes, and MCAs. Figure 10 of Appendix A contains the results of the Block I/O Throughput Tests.

(j) Shared Page Re-Map Throughput Test. This test operated as a single process with two tasks. One task performed initialization and measurement period control, and the second task performed ?SPAGE system calls inside a tight loop. The ?SPAGE call is important because it is the basic element used in performing the virtual overlay function within AOS and ARTS. All CMS-2M and FORTRAN programs in our systems use the ?SPAGE mechanism when performing procedure calls and subroutine calls to procedures and subroutines which have been bound as overlays. The output of this test is a count of ?SPAGE calls which can be executed per second. The test allowed the number of pages read by each ?SPAGE call to be varied so that the system response can be measured as a function of the number of pages being read. Figure 11 of Appendix A summarizes the results of the Shared Page Re-Map Test.

At the time of writing of this report, the side-by-side system execution tests had not been completed, due to 4C scheduling conflicts. Subjective observations are that responsiveness is excellent with ARTS, but quantitative measurements are not available (see Figure 12).

6. SUMMARY

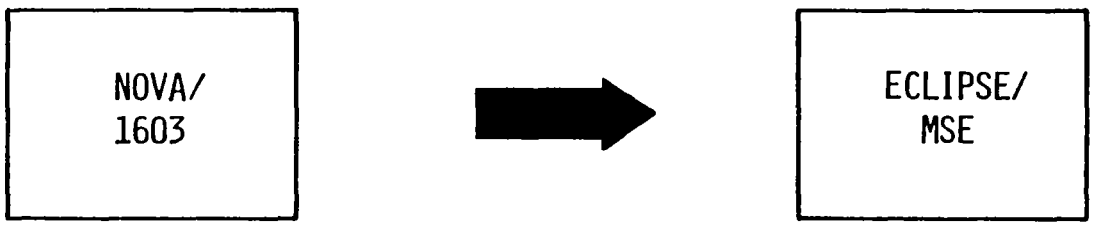
4C feels that the Advanced Real Time System definitely meets the 4C goals for a real time replacement for AOS. ARTS is faster, smaller, and more configurable than AOS, and it is compatible with AOS to a very large extent. The areas of incompatibility are limited to those functions which are not a part of the projected ARTS environment, and once these differences were understood, we had few problems working and operating within the ARTS capabilities. 4C feels that the existence of ARTS is very complementary to AOS, and it naturally lends itself to an ideal program development and checkout environment: programs can be developed under AOS, debugged under AOS, and then installed under ARTS for final checkout and delivery. Because ARTS is operable on both MIL-SPEC and commercial ECLIPSE computer systems, it is possible to develop both commercial and MIL-SPEC versions of a system and have them execute exactly the same programs. Finally, the close compatibility between ARTS and AOS means that our software investments in applications software, the CMS-2M compiler, and other software utilities are preserved.

In conclusion, 4C is happy to recommend ARTS to the ROLM and Data General computer communities, and 4C hopes to use ARTS in its own products as soon as possible.

APPENDIX A

ARTS BETA TEST REPORT BRIEFING CHARTS

EVOLUTION OF REQUIREMENTS



- 1 PROCESS
- MEMORY RESIDENT
- ASSEMBLY LANGUAGE
- FIXED FUNCTION
- FIXED POINT

- MULTIPLE PROCESSES
- PRIMARILY MEMORY RESIDENT
- CMS-2, FORTRAN, ASSEMBLY MIX
- FIXED AND VARIABLE FUNCTIONS
- FIXED POINT/FLOATING POINT MIX

Figure 1

TYPICAL LARGE SYSTEM CONFIGURATION

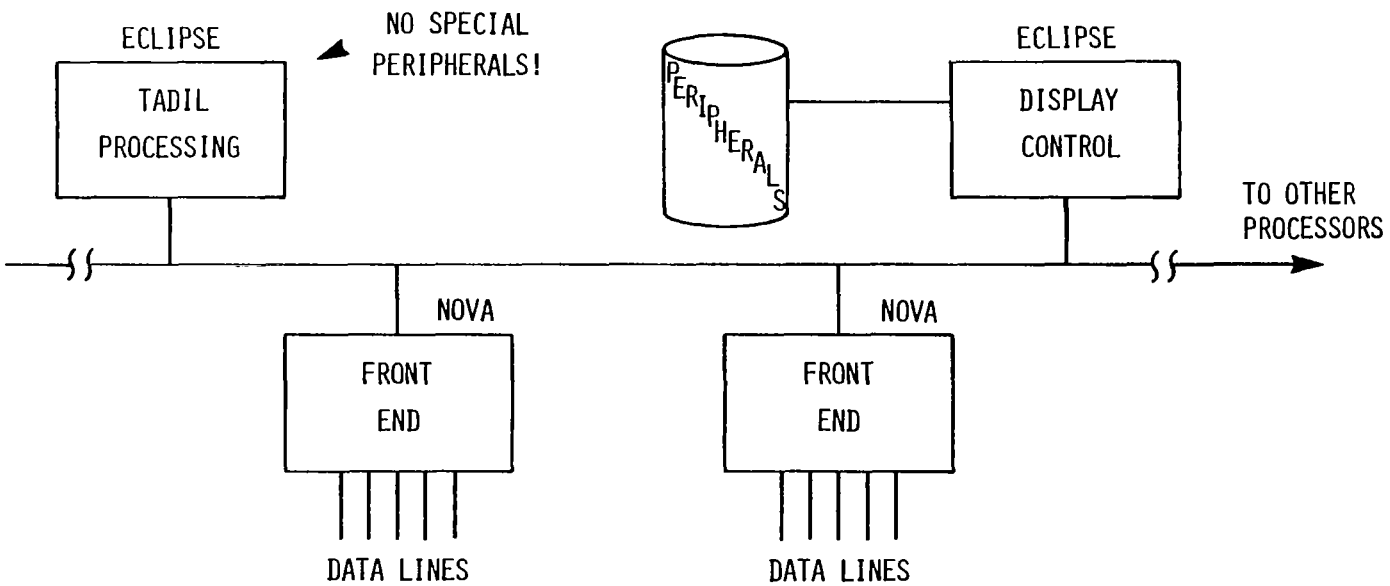


Figure 2

AOS HARDWARE ENVIRONMENT

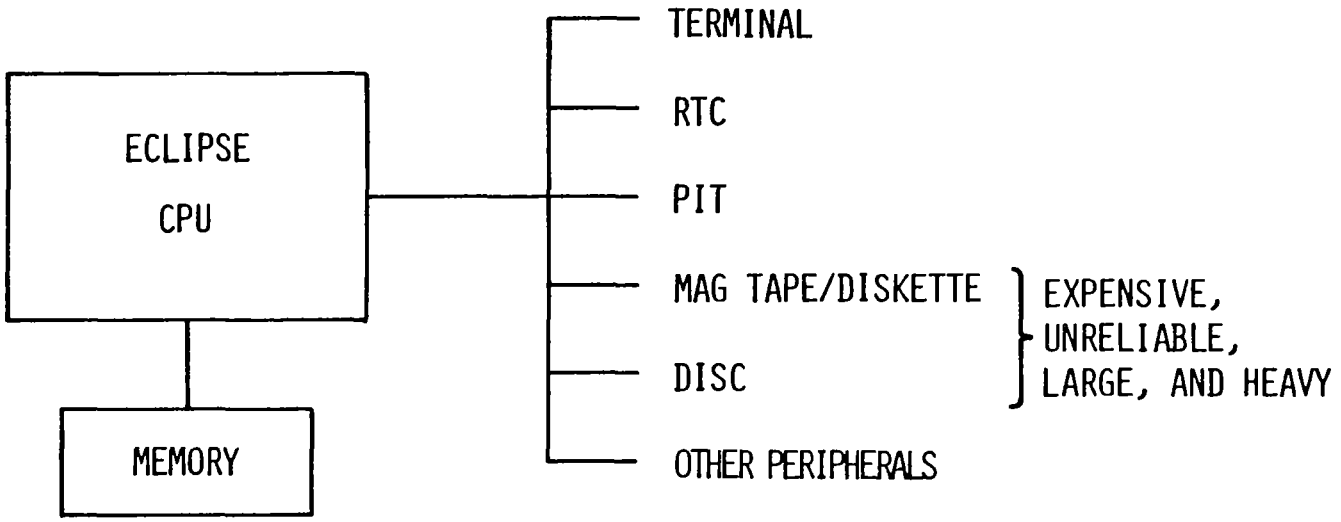


Figure 3

FEATURES DESIRED IN A REAL TIME AOS

- AOS COMPATIBILITY
  - CODE + DATA STRUCTURES
  - SYSTEM CALL
  - PROGRAM FILE (NO RE-BIND NEEDED!)
- MEMORY RESIDENT
  - SYSTEM KERNEL, SYSTEM OVERLAYS, AND SYSTEM DATA
  - GHOST + GHOST OVERLAYS
  - PMGR
  - APPLICATIONS PROGRAMS, OVERLAYS, AND DATA
  - IPC SPOOL FILES
- REAL TIME RESPONSE
  - LOW OVERHEAD
  - FAST SYSTEM CALL PROCESSING
- MINIMAL I/O PERIPHERAL REQUIREMENTS
  - DISK ONLY
  - TAPE ONLY
  - MCA ONLY
  - NONE OF THE ABOVE

Figure 4

INITIAL RESULTS

- ARTS INSTALLATION TEAM WAS ON SITE FOR 4 DAYS
- ARTS WAS INSTALLED AND CONFIGURED (BY 4C) IN ONE DAY
- ROLM AND 4C PERSONNEL HAD A 6-PROCESS, REAL TIME SYSTEM RUNNING WITHIN 4 DAYS. SYSTEM INCLUDED:
    - GRAPHIC DISPLAY
    - TACTICAL DATA LINK (TADIL-B)
    - DATA COLLECTION/REDUCTION
    - 4 "IDEF" DEVICES ACTIVE
- INITIAL PROBLEMS WERE:
    - NONSUPPORTED CALLS (?CREATE)
    - CODING ERRORS (APPROXIMATELY 10 WERE FOUND)

Figure 5

COMPATABILITY/INTEGRITY TEST RESULTS

- EXISTING FUNCTIONAL TEST WAS DEVELOPED FOR VERIFYING OPERABILITY OF A 4C PROPRIETARY NETWORK TRANSACTION PACKAGE WHICH SUPPORTS
    - INTER-TASK COMMUNICATIONS
    - INTER-PROCESS COMMUNICATIONS
        - IPC
        - SHARED FILES/PAGES
    - INTER-CPU COMMUNICATIONS
        - MCA
- INTER-TASK COMMUNICATIONS WORKED IMMEDIATELY
- INTER-PROCESS AND INTER-CPU COMMUNICATIONS TESTS FOUND CODING ERRORS
- ALL TESTS ARE NOW FULLY OPERATIONAL

Figure 6

SYSTEM OVERHEAD MEASUREMENTS

$$\text{SYSTEM OVERHEAD} = \frac{\text{MAX-MEASURED}}{\text{MAX}} * 100$$

$$\text{AVAILABLE TIME } (\mu\text{SEC}) = \frac{\text{MEASURED}}{\text{MAX}} * 10^6$$

CLOCK RATE	LOOPS AOS	LOOPS ARTS	PERCENT OVERHEAD AOS	PERCENT OVERHEAD ARTS	AVAILABLE TIME AOS	AVAILABLE TIME ARTS
10HZ	80050	107236	25.4%	0.087%	745,836	999,134
100HZ	76961	106965	28.3	0.34	717,057	996,608
1000HZ	45292	102365	57.8	4.6	421,992	953,750

(MAXIMUM LOOPS = 107,329)

Figure 7

SCHEDULING OVERHEAD RESULTS

SCHEDULING OVERHEAD IS DETERMINED BY CHANGE IN AVAILABLE PROCESSING (IDLE) TIME WITH 1, 2, OR 3 BACKGROUND TASKS RUNNING.

NUMBER OF PROCESSES	AOS 10HZ	AOS 100HZ	AOS 1000HZ	10HZ	100HZ	1000HZ
1	2.95%	27.1%	-	1.73%	17.0%	82.9%
2	5.44	49.4	-	3.12	31.0	-
3	7.65	-	-	4.54	45.1	-
AVERAGE/ PROCESS	2.67	25.5		1.56	15.5	

AOS RESCHEDULE $\cong$ 2600 μSEC

ARTS RESCHEDULE $\cong$ 1550 μSEC

Figure 8

THROUGHPUT TEST RESULTS

CALL/TEST TYPE	TIME PER CYCLE (µSEC) AOS	ARTS
IPC	10968	8764
?XMT	81	75
?REC	74	67
?XMT/?REC	726	1022*

* - SUSPECTED ARTS CODING ERROR

Figure 9

I/O TEST RESULTS

- CHARACTER I/O OUTPUT TO "@CON0"

  AOS = 783 CHARACTERS/SECOND
  ARTS = 830 CHARACTERS/SECOND

- FILE I/O (OPEN, READ 50 RECORDS, CLOSE)

  AOS = 80 CYCLES/MINUTE
  ARTS = 266 CYCLES/MINUTE

- BLOCK I/O OF 1 WORD TO/FROM AN MCA

	AOS	ARTS
?WRB	2880 µSEC	3000 µSEC
?RDB	2825 µSEC	3000 µSEC

Figure 10

SHARED PAGE RE-MAP TEST RESULTS

NUMBER OF PAGES "READ"	TIME PER CALL AOS	ARTS
1	1551 μSEC	1332 μSEC
2	2187	1705
3	2793	2072
4	3421	2443
8	5873	3934
12	8380	5430
16	10968	6891

Figure 11

SYSTEM PERFORMANCE TEST RESULTS

- THESE TESTS ARE IN PROGRESS BUT INCOMPLETE
- SUBJECTIVE OPINIONS THUS FAR:
  - ARTS IS PERFORMING AS EXPECTED
  - ARTS START-UP/RESTART TIME IS AN ORDER OF MAGNITUDE LESS THAN THAT OF AOS
  - WE ANTICIPATE A 30% + IMPROVEMENT IN SYSTEM THROUGHPUT

Figure 12

REAL TIME SOFTWARE TOOLS AND METHODOLOGIES

M. J. Christofferson
E-Systems Inc., Melpar Division
Falls Church, Virginia

ABSTRACT

In designing software for a real time processing system of any complexity, the software analyst is presented with a wide variety of design choices and software structures to use. Real time systems are often characterized by high speed processing and throughput as well as asynchronous event processing requirements. These requirements give rise to particular implementations of parallel or pipeline multitasking structures, of inter-task or inter-process communications mechanisms, and finally of message (buffer) routing or switching mechanisms. These mechanisms or structures, along with the data structure, describe the essential character of the system.

This paper reports on attempts by the author and his co-workers to isolate these common structural elements and mechanisms and formalize their implementation in the form of routines, tasks or macros — in other words, tools. The tools which have been developed support or make available the following:

— Re-entrant task creation
— Generalized message routing techniques
— Generalized task structures/task families
— Standardized inter-task communications mechanisms
— Pipeline and parallel processing architectures in a multi-tasking environment

Tools development as discussed above raises some interesting prospects in the areas of software instrumentation and software portability. These issues will be discussed following the description of the tools themselves. The tools described have been specifically developed for a ROLM 1666 under RMX/RDOS.

This paper describes the set of software tools developed at Melpar to facilitate the design and implementation of real time software systems. It appears some of these tools address problems which are generic to the development of real time software systems, and therefore of general interest. Some preliminary statements must be made prior to proceeding to a description of the software tools sets.

The software systems in which these tools are used are characterized by their event driven nature, their high speed I/O and processing requirements, and frequent severe restrictions of processor size and weight. The functional requirements of these systems demand real time, multi-tasking architectures. No adequate higher order language is available which specifically addresses the issues pertinent and generic to real-time system architectures. This coupled with the memory restrictions associated with size and weight constraints resulted in a decision in favor of Assembly language.

There are, of course, many notorious disadvantages to Assembly language programming, which by itself does not really address some of the critical issues of real time programming. The solution to this dilemma is to develop software tools, constructs, or mechanisms specifically designed to address these issues.

The tools developed at Melpar fall into four major categories. The first of these is the source language constructs, which consist of a library of macros that provide some of the features of higher order, structured programming languages. The second category consists of a library of generalized memory management routines, which perform buffer allocation, linked list access, and similar functions. The third category consists of a variety of task creation and inter-task communications constructs. The fourth category consists of generalized multi-tasking and process communications structures. These four categories suggest a hierarchy, and indeed they are listed in order of development. Furthermore, each successive category is built on its predecessors.

As mentioned above, the first level of the software tools hierarchy consists of source language constructs. These are a collection of macros which provide some of the control structures offered by higher order languages such as PASCAL. These structures include IF-THEN-ELSE, WHILE (boolean), DO, DO-UNTIL (boolean), CASE, and FOR loop constructs. It is not the intention of this paper to describe these lower level tools in detail. Suffice to say that they greatly assist the programmer to produce, at the Assembly language level, well-structured, maintainable and easily modified code. In short, these tools are designed to overcome some of the many disadvantages of Assembly language programming.

An example of a program which employs these source language tools is given in Figure 1. It should be stated that these macros resolve the relocation properties (register, absolute,relocatable, external, and so forth) of the symbols referenced in arithmetic and boolean expressions. Any level of nesting is allowed, as is any properly formed boolean or arithmetic expression. Furthermore, all of these macros are completely re-entrant.

The three higher levels of the software tools hierarchy may be considered as system programming and design tools. Tausworthe (ref. 1) states that "Real time programming efforts are dominated by the human incapability to comprehend the total picture of what is really going on in the computer on an instant by instant basis." One of the most important roles of the higher level software tools is to add to the mental set of the system designer, providing a language with which to describe real time architecture and mechanisms. It has been our experience that the availability of these tools has indeed been a great boon to our systems designers, partly because they address the problem mentioned by Tausworthe.

There are two ideas which have driven the development of the higher level tools. The first idea was to develop

and formalize, where possible, general software mechanisms for handling commonly employed real time, multi-tasking system functions. The second idea was to treat these standardized mechanisms as an augmentation or extension of the resident operating system of the machine. In this manner, the mechanisms become building blocks with which to design real time software systems.

The approach taken in the development of these constructs is reflected in the hierarchy mentioned before. First, generalized buffer and memory management utilities were developed. These utilities include buffer pool construction routines, buffer allocation and de-allocation routines, and linked list management routines. Not much need be said about these, as such utilities are fairly common.

The next step in the ascending hierarchy consists of the inter-task communications and task spawning tools. These tools enhance the system tasking and inter-task communications mechanisms of the operating system, using a macro approach.

The final layer of the software tools hierarchy builds upon the two previous layers, and consists of highly generalized multi-tasking and inter-process communications structures. Our description of these last two levels of

the hierarchy begins with task communications and spawning tools.

These tools were designed to support re-entrance at the task level and standardize common inter-task communications mechanisms. They are transportable from system to system, being completely independent of application. The tools in this level of the hierarchy use the macro assembler facility to standardize the following functions:

1) task or process creation
2) queueing of a buffer and transmission to another process
3) de-queueing of a buffer sent by another process
4) FORK-JOIN structures for concurrent processes.

The task creation tool is known as the CTASK mechanism. CTASK allocates a buffer from the buffer pool and passes the address of the buffer to the created task via the data link. This buffer is used as a stack area for the created task. In this manner, re-entrant tasks are easily created. Furthermore, any number of arguments may be inserted into the pool buffer in a pre-determined order prior to task creation. In this manner, when the task is created, any number of arguments may be passed to it on the stack buffer.

The format of the CTASK call is as follows:

CTASK ENTRY ID PRIORITY DATALINK STACKSIZE <ARGLIST>

The arguments ENTRY, ID, PRIORITY and DATALINK are arguments supplied to the .TASK system call. The remaining arguments, including the optional argument list, are peculiar to the CTASK mechanism. STACKSIZE is of course the size of the stack buffer to be passed to the task through the data link. Any optional arguments are placed on the stack buffer, as illustrated in Figure 2.

The created task first calls the macro ITASK, which requires no arguments. ITASK initializes the stack registers and extracts the user-specified DATALINK from the stack and places it in register 2. The KTASK macro is called when the task is ready to remove itself from the system, and causes the stack buffer to be returned to the pool prior to issuing a .KILL task call.

As we shall later see, CTASK, ITASK and KTASK play an important role in the development of subsequent structures.

Inter-task communications are facilitated by the QREC and QSEND macros. QSEND links a buffer to a specified linked list and transmits, via the .XMT task call, a message to ready the receiving task, if suspended. QREC

is the macro called by the receiving task to accept the transmitted buffer from its linked list. QREC searches the indicated prioritized linked list for the highest priority non-empty list and retrieves that buffer. If all the lists are empty, then the task will suspend on a .REC task call until a subsequent QSEND operation is invoked by another task. Figure 3 illustrates the QSEND/QREC mechanism as it operates.

Under development are tools for implementing FORK-JOIN structures for concurrent processes. FORK-JOIN structures are discussed in detail by Tausworthe (ref. 1) and the concept is illustrated in Figure 4. However, as yet the details of syntax and implementation have not been completely worked out. In all probability, the FORK-JOIN mechanism will use the CTASK macro as a sub call to create independent processes.

It is now time to address the last level of the software tools hierarchy, which consists of generalized multi-tasking and inter-process communications structures. The generalized multi-tasking structures shall be described first.

The generalized multi-tasking structures were developed to support tasks of commonly encountered types. Thus far three major "families" of tasks have been defined. The

first family is best described as consisting of permanent tasks which are driven by the receipt of buffers on a linked list. The second family of tasks are temporary tasks, which perform a function and then perform a KTASK operation. Since TCB's are by no means a super-abundant system resource in most operating systems, a limited task spawning feature has been associated with temporary tasks. This permits a prioritized throttling process which governs task creation, thus preventing any attempt to spawn an excessive number of tasks during periods of high system activity. The third task family is the concurrent process task structure. These tasks will be created by the as yet undefined FORK-JOIN mechanism mentioned before.

These task structures are implemented as task "shrouds" which call one or more embedded "personality modules." The task shroud contains the general-purpose code which executes the function of the applicable task family. This code is table driven by parameters passed to the task at creation time via the CTASK mechanism. The address of the personality module is itself one of the arguments passed to the task at creation time. These structures are completely re-entrant, thereby permitting several applications modules to be served by the same task level support code. The task shroud code uses the task communications and memory management structures as integral building blocks.

Figure 5 illustrates the basic structure of the three defined task shrouds. We shall choose one of these shrouds as an example for discussion.

The re-entrant task shroud TPFLL is designed to support a permanent task which is driven by receipt of buffers on an input linked list. Therefore, the task must perform a QREC operation, retrieving a buffer from the linked list specified by the creation argument list. It must then call the specified personality module, passing this buffer to that routine. Upon return from that routine, it must loop back to the QREC call to retrieve the next input buffer. Obviously, these are the minimum functions which a task of this type must perform. Other generalized processing functions may be associated with tasks of this family, and duly inserted in the code of the shroud. For example, one may wish to call a routine once upon initialization of the task. The critical notion behind the concept of a task shroud is, of course, that once a generic family has been identified (and coded), this structure may be used as a building block by which to implement the architecture of the system. One obvious advantage of this is that the code need not be reproduced. It also addresses the problem described by Tausworthe (ref. 1) in it adds to the conceptual vocabulary of the systems designer.

Examples of the limited spawning and concurrent processing (FORK-JOIN) task shrouds are also illustrated in Figure 5.

Figure 6 illustrates skeleton architecture of a hypothetical system, using all the task shroud families currently defined. The diagram consists of several blocks which represent tasks. The task family name appears in the upper portion of the block, and the associated personality module in the lower portion. The figure clearly illustrates the separation of the software system architecture from the system functional architecture. The former is represented by the task shrouds and the control and data connectives between them. Notice that this representation is not dependent upon the specific details of the applications-specific software. The system functional architecture is represented by the group of personality subroutines. The systems designer can easily construct the software architecture, and turn his attention to the design of the functional architecture of the system. The applications programmer now needs only to write subroutines, which can be debugged in place rather than in some hastily contrived test frame. This dramatically improves performance in both debug and system integration phases.

We will now turn our attention from generalized multitasking structures to generalized inter-process communications

constructs. The most interesting of these is the construct called ROUTER. ROUTER is simply a re-entrant subroutine package which makes it possible for a process to communicate with another process on a logical basis, without explicit knowledge of the physical communications path. This is accomplished by defining a data structure called the ROUTER table. This is a parallel table, containing the information required to establish the physical communications path (e.g., linked list header address, message cell address, subroutine address, etc.). A particular communications path is identified by a so-called "logical unit" number, which is merely an index into this parallel table pointing to the appropriate communications path. The source process passes to ROUTER the logical unit number of the destination process and the data to be transmitted. The concept of the ROUTER mechanism becomes especially useful in consideration of modular design and software transportability requirements.

The thrust of this paper has been to show that the construction of generalized software tools can aid in both the design and implementation of real time software systems. Several constructs have been outlined above which we believe have general application in real time systems. However, it is worth emphasizing once again that much of the value associated with tools definition and usage is due to the enhanced software design vocabulary which these tools provide.

REFERENCE

1. Tausworthe, R. C.: Standardized Development of Computer Software. Prentice-Hall, 1977.

```
                ;//////////////////////////////
                ;
                ;     READY TASKS THAT ALWAYS RUN
                ;
                ;//////////////////////////////

                CRTSK:
000441'075101           PSH     R3                      ;SAVE RETURN ADDR
000442'071101           PSH     R2                      ;PRESERVE REGISTERS
000443'065101           PSH     R1
000444'061101           PSH     R0

000445'100010           LEF     R0,ITATS                ;STARTING ADDR OF TABLE IN R0.
        177777
000447'114010           LEF     R3,ITAID                ;ADDR OF NEXT COLUMN
        177777
                        SET R3 = R3 - R0                ;LENGTH OF TABLE = NTASKS -1
                        SET R0 = 0                      ;INITIALIZE ROW COUNTER
                        XSET R1 = ITATS < R0 >          ;JUST TO GET STARTED
                      WHILE ( R1 NE 0 )                 ;MAIN LOOP
                        IF ( R1 NE -1 )
                            XSET R1 = NPTR < R0 >       ;PTR TO OPTION ARGUMENT LIST
                            XSET R2 = R1 < 0 >          ;NUMBER OF ARGS IN THIS LIST
                            SET R1 = R1 + R2            ;POINT TO LAST ARG IN LIST
                            XSET R2 = NPTR < R0 >
                          WHILE ( R2 NE R1 )
                              XSET R2 = R1 < 0 >        ;R2 <- ARG
000511'071101                 PSH R2                    ;PUSH ARG ONTO THE STACK
                              SET R1 = R1 - 1           ;POINT TO PREVIOUS ARG IN LIST
                              XSET R2 = NPTR < R0 >
                          ENDWHILE
                            ;PUSH MANDATORY ARGS ONTO STACK
                            ;INDEXING ACCROSS THE TABLE
                          FOR R2 = 0 TO 4
                              SET R1 = 4 - R2
                              SET R1 = R1 * R3 + R0
                              XSET R1 = ITATS < R1 >
000550'065101                 PSH R1
                          ENDFOR
                            XSET R2 = NPTR < R0 >
                            XSET R2 = R2 < 0 >
000561'071101               PSH R2                      ;PUSH NUMBER OF EXTRA ARGS
                            SET R1 = R3     ;PRESERVE R3
                            CALLP C.TASK < GOTO ERTSK >
                            SET R3 = R1     ;RESTORE R3
                        ENDIF
                        SET R0 = R0 + 1
                        XSET R1 = ITATS < R0 >
                      ENDWHILE
                        GOTO LEXT
                ERTSK:  CALLJ   ERINT
                LEXT:
000606'061201           POP R0          ;RESTORE CONTEXT
000607'065201           POP R1
000610'071201           POP R2
000611'060201           PRT
```

242

TOTAL CODE

150_8

Figure 1

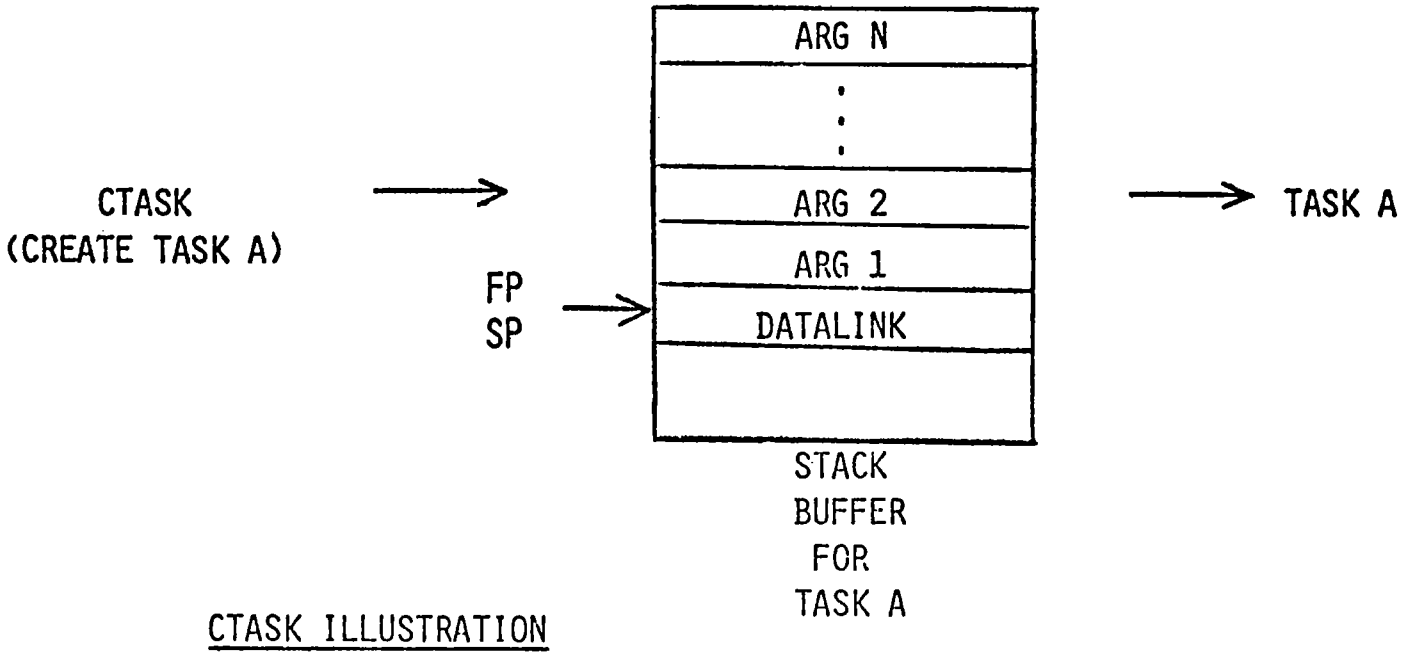


Figure 2

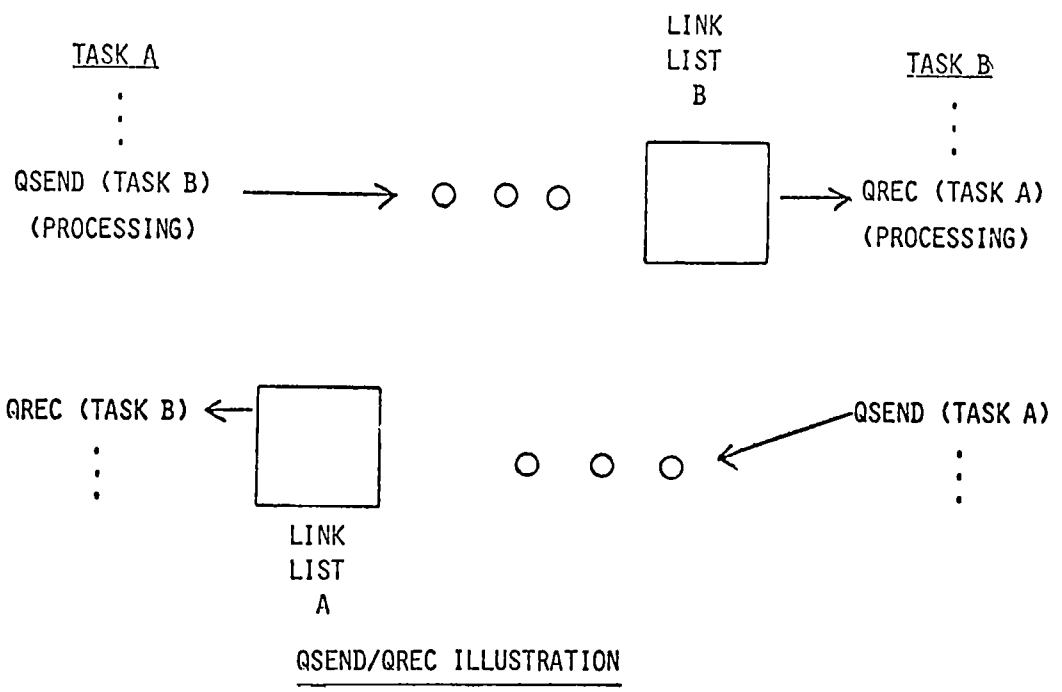
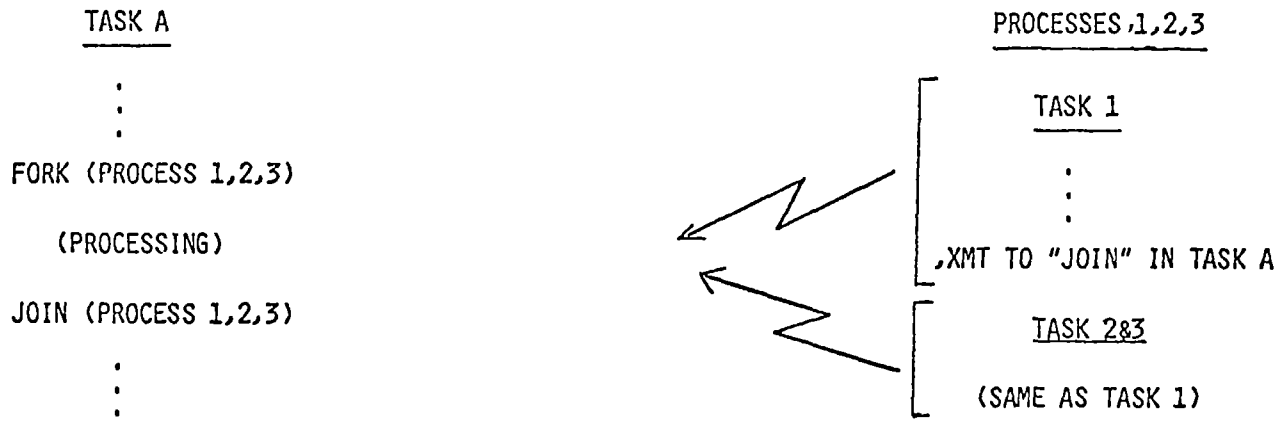


Figure 3



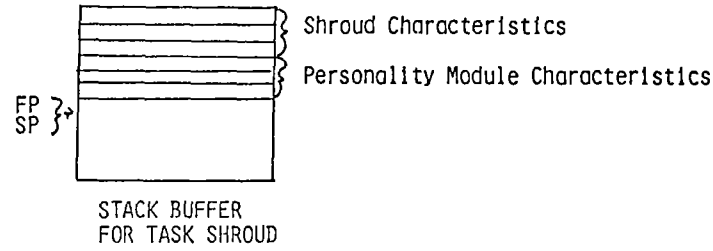
FORK-JOIN ILLUSTRATION

QUESTION: WHAT IS A CONVENIENT MECHANISM FOR PROCESSES 1,2,&3 TO COMMUNICATE WITH THE "JOIN" IN TASK A?

ANSWER: WE'LL SEE LATER.

Figure 4

GENERALIZED MULTITASKING CONSTRUCTS



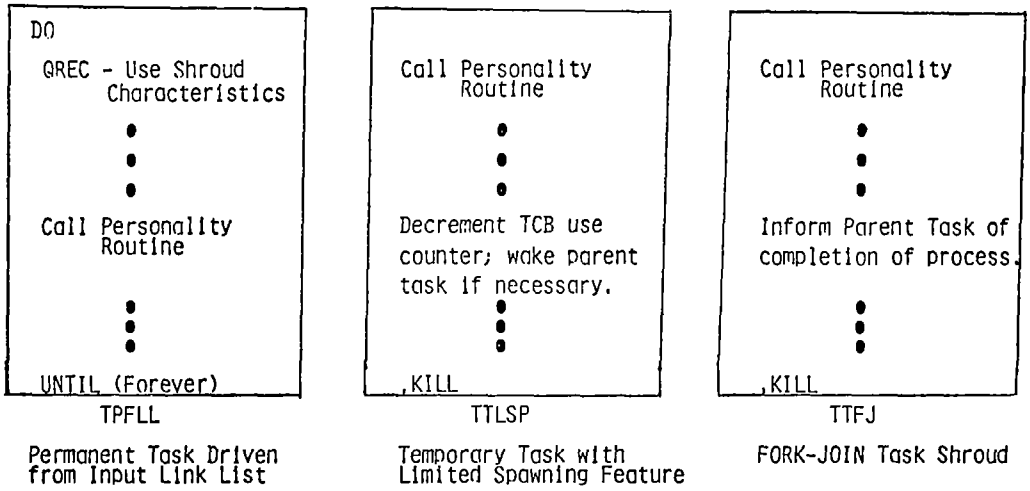


Figure 5

HYPOTHETICAL MULTITASKING ARCHITECTURE

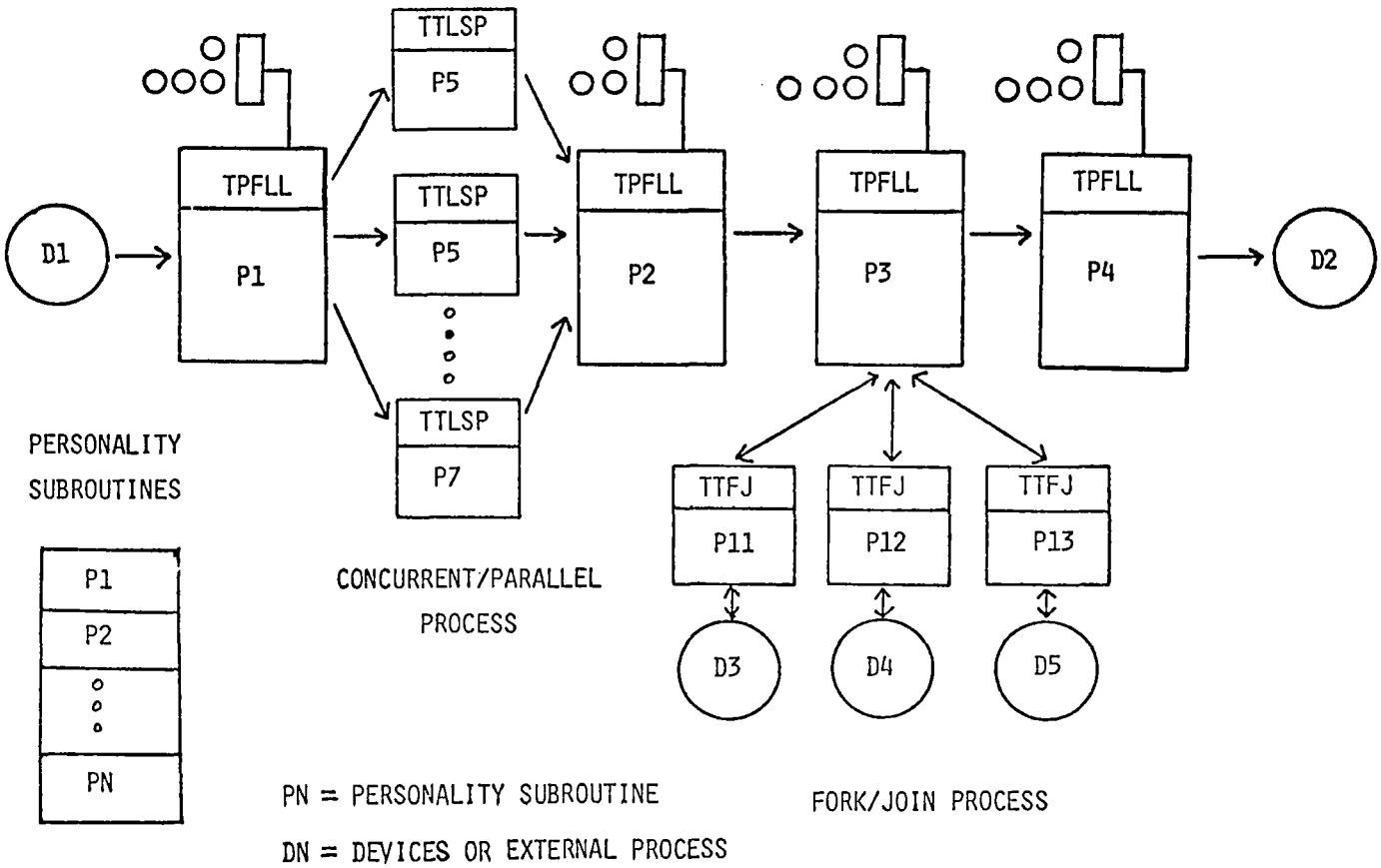


Figure 6

USE OF SOFTWARE TOOLS IN THE DEVELOPMENT OF REAL TIME SOFTWARE SYSTEMS

Robert C. Garvey
E-Systems Inc., Melpar Division
Falls Church, Virginia

ABSTRACT

This paper will discuss the transformation of a pre-existing software system into a larger and more versatile system with different mission requirements. The history of this transformation is used to illustrate the use of structured real-time programming techniques and tools to produce maintainable and somewhat transportable systems.

The predecessor system, which is called SE, is a single ground diagnostic system. Its purpose is to exercise a computer controlled hardware set prior to its deployment in its functional environment, as well as test the equipment set by supplying certain well known stimulae. The successor system, called FTF, is required to perform certain testing and control functions while this hardware set is in its functional environment.

Both systems must deal with heavy user I/O loads and a new I/O requirement was included in the design of the FTF system. Human factors were enhanced by adding an improved console interface and special function keyboard handler. The additional features required the inclusion of much new software to the original set from which FTF was developed. As a result, it was necessary to split the system into a dual programming configuration with high rates of inter-ground communications. A generalized information routing mechanism was used to support this configuration.

The architectures of the two systems will be presented briefly. The remainder of the paper will describe the use of the software tools and techniques discussed by Mr. M. J. Christofferson in performing this upgrade. Special emphasis is placed on the utility of such tools in a system upgrade effort. The issues of increased programmer productivity and maintainability of software are also addressed.

Christofferson (ref. 1) has presented the set of software tools developed at Melpar to facilitate the design and implementation of real time software systems. This paper is offered as a commentary on the software tools set. It is our intent to illustrate the value of these tools with a case example and a discussion of some quantitative job performance measures.

The software tools set may be divided into two categories. The first of these may be referred to as "code level macros" and the second as "architectural level tools and utilities."

The first category consists of the Logical Constructs Macros and the Data Structures Macros, which were designed to overcome some of the many disadvantages of Assembly language programming.

The second category is of more general interest, as these tools represent functions which are believed to be generic to most real time software systems. These tools include the generalized inter-task communications software, which includes the information routing mechanism described by Christofferson, and the generalized multi-tasking structures. They are supported by a host of system level utilities which include memory management, linked list access and other general-purpose routines.

Christofferson discusses in his paper the notions of separating applications-specific code from that code which expresses the architecture of the system, of insulating the architectural level and applications-

specific code somewhat from the peculiarities of the operation system, and of establishing logical rather than physical data and control connectives between modules. The first of these three general philosophies finds its expressions in the general multi-tasking structures package, which consists of a collection of commonly applicable task shrouds. The second notion finds its expression in the generalized inter-task communications tools.

One of these tools is of particular importance to the case example which follows these introductory remarks. This is the information routing mechanism, or ROUTER. This tool allows a task communication entity, such as a message cell or linked list header structure, or a subroutine to be referred to on a logical rather than physical basis. It is both a generalized inter-module communications mechanism and an expression of the philosophy which states that data and control connectives between modules should be logically, rather than physically, established within the communicating modules. This is achieved by establishing a data structure called the Router Table. The Router Table contains the information required to translate the logical to the physical connection. The mechanics of performing this translation are the responsibility of the ROUTER software, not the user. The user of ROUTER refers to the entity with which communication is desired by a logical unit number which identifies a particular set of physical characteristics associated with the logical device.

In our work we have discovered that this ability to establish logical, rather than physical, data and control connectives is extremely useful in the implementation of structured systems designs. In particular, adherence to this philosophy makes it practical to extract a software subsystem from an existing system and drop it into the architectural

framework, or "skeleton," of a new system. The new system, of course, may fulfill a completely different set of functional requirements; the inserted module merely fulfills a system level function required in both. This is a fairly common practice in our firm, and since re-invention of the wheel is never popular, no doubt it is also the case in most other firms. The case example presented later in this paper deals with a project where vast amounts of code were transferred from an existing system to a new system.

It is our claim that adherence to such philosophies greatly benefits the system development team, from cost estimator to applications programmer.

It is impossible for the cost estimator to know all the ins and outs, the little quirks and interfacing problems, which may be characteristic of any given module which may be used in the new system. There is, therefore, a great deal of uncertainty when the estimator tries to assess the cost of installing the module in the environment of the new system. The software tools reduce this uncertainty by a significant percentage for the following reasons.

First, the estimator may be assured that the module was implemented in accordance with the overall design philosophy with which he or she is familiar. Second, the job estimator knows that the architecture of the system need not be considered sacrosanct. The software tools allow rapid modifications of the skeleton system, including the easy addition of new tasks and the logical connectives between them and the rest of the system. Thus, if for some reason the original architecture proves to be undesirable, a new architecture can be designed and implemented without great delays in meeting program requirements. Our case example describes a situation where this kind of problem did in fact occur,

and re-design of the skeleton system did in fact prove to be the most efficient solution.

To a systems designer, the software tools provide a group of constructs that can be used to lay out the architectural level of the system. The architectural configuration, or skeleton, of the system is, of course, dependent on the application. The software tools allow the designer to conveniently define the mechanisms by which information arrives at the proper module in the proper format at the proper time. The designer knows that because of the availability of the tools, the design can be implemented in a clean fashion which truly represents the structure laid out in the design process. The designer knows that the skeleton system can be quickly constructed and provided to the applications programmer as a test frame for the applications-specific software.

This approach greatly diminishes debug and integration labor requirements and frees the applications programmer to concentrate entirely on the application at hand. Initial debug efforts are performed in what is growing into the operational environment of the applications-specific software. Inter-task and module communications mechanisms do not have to be constantly debugged each time a new task is installed.

These claims will be demonstrated by means of a case example, which we shall now introduce.

HISTORICAL BACKGROUND

Our firm's experience with ROLM computers began in 1977. The first system to be built by Melpar using ROLM computers was a multiprocessor system, based on the ROLM 1650. This system was designed to control

and route information from a set of hardware devices to a remote installation.

Our second ROLM based system was established on a 1666 processor. Its function was to perform diagnostics and fault localization on the hardware set controlled by System 1 while in a laboratory environment.

The third system was designed to meet a completely different set of functional requirements. Unlike Systems 1 and 2, it was not characterized by large amounts of user I/O, but was required to perform a great deal of computations on data as it arrived.

By the time System 3 was delivered to its customer, System 2 had performed in the field for quite some time and was regarded as a success. Indeed, the customer noted the need for a similar system in a different phase of its operations with System 1. System 4 was conceived to supplement the ability of a pre-existing system to control System 1, as well as perform remote diagnostics and fault localization while System 1 is deployed in its operational environment.

System 4 was, therefore, required to perform all the diagnostic functions of System 2, as well as receive commands which it could translate and route to System 1. No longer could operation be performed over a cable. Communication between System 1 and System 4 was to be over a remote link.

Figure 1 illustrates the evolution of the software tools hierarchy against the development of these four systems. The programming aids and data structures were available to System 1 programmers only rather late in the development, and were a by-product of the efforts of the programmers to produce a clean implementation. System 2 programmers had the benefit of general utilities developed in the early stages of that project such as memory management routines and linked list access

packages. System 3 programmers developed the inter-task communications tools as part of the effort to implement a clean ground-to-ground communication scheme, and used them extensively. The general multi-tasking structures were also developed during this project, but due to their lateness were not extensively used.

System 4 was, therefore, the first project to have the advantage and the availability of the complete hierarchy of software tools.

CASE EXAMPLE

As mentioned before, the customer saw the need to expand the capabilites of System 2 and place that system in another operational environment. The conversion of System 2 to meet these new functional requirements provides the topic for this case example. The resulting product of the conversion and upgrade effort shall be referred to as System 4. It is important to note that the missions of the two systems are similar, but with important differences.

Figure 2 is a much simplified block diagram of System 2. I/O handlers, protocol handlers and the like are omitted because little difficulty was experienced in transforming these modules to meet System 4's requirements. Notice the blocks labeled METAI and OPTSK. The interaction of these two task level modules is of particular importance to this case example.

The METAI task is basically a command interpreter. It accepts commands in the form of ASCII strings and produces binary coded packed buffer structures called plexes. These packed buffer structures are transmitted to the task labeled TEST EXEC, which interacts with the task labeled SEQUENCER to provide synchronization and sequencing control over the execution of the operational tasks.

OPTSK is a re-entrant task shroud which is created as needed by SEQUENCER to execute the command currently being processed. In general, any given number of OPTSK's may run at any given time. OPTSK performs a vectored subroutine call to a given operational routine which executes a given command. Again, we see the concept of separating the applications-specific software, the operational routines, from the skeleton or architectural layer of the system.

Note that OPTSK accesses a subroutine package which is also used to support the operation of the command interpreter. This was done so that looping capability could be provided; the OPTSK would extract parameters from a disk file and fill in the proper areas of the command block plex with appropriate parameters and execute the command once for each parameter in the file. It is this feature which was to cause so many problems in the development of System 4's skeleton.

At this point, it should be pointed out that the command interpreter was implemented in a manner which egregiously violated the design philosophy indicated in previous portions of this paper. Physical, rather than logical, data and control connectives were established between the interpreter and the supporting package of buffer formatting routines. Furthermore, use of packed buffer structure access tools was not consistantly applied throughout these modules, making it difficult to change the format of the output buffer header. Such a change was required to meet the overhead requirements of the ROUTER.

System 4's original architecture is represented in Figure 3. This is again a very much simplified block diagram which concentrates on the interaction between the command interpreter and the operational task. Note that the buffer formatting routines were to be resident in both grounds.

It was decided to first tackle the problem of including the router header space in the definition of the plex , and establish a single ground system to test compatability with the rest of the system. This involved modifications with existing software so that METAI would now communicate with the Test Executive via means of the ROUTER facility, a fairly simple configuration to arrange. This required about two weeks of effort, most of which was devoted to implementing the appropriate modifications in the code of the interpreter.

It should be pointed out at this time that the programmer assigned to the task of skeleton system construction was an entry-level employee with no experience in real time systems. This indicates the confidence of the project management in the tools, not necessarily in the employee.

This single ground system was delivered to the applications programmers as an initial test frame and work on establishing the two ground skeletons began.

The skeleton represented in Figure 3 was established in one man-month of effort. Under most conditions, it worked well, but it soon became apparent that the looping capability mentioned earlier had not been achieved. Futher examination of the software showed that the buffer formatting routines were utterly dependent upon data and code spaces within the command interpreter.

The only alternatives were to rearrange the architecture of the system, or commit the project to the task of performing major surgery on the offending modules. After due consideration of the size and complexity of the offending modules, it was decided to resort to the former alternative.

Several skeletons were implemented over the next month in an attempt to find an efficient way around the problem. At the end of that period of time, the architecture of Figure 3 has been transformed to that suggested by Figure 4. Note that the ground-to-ground split was not established between METAI and the TEST EXEC as before; it is the operational task that has become the point of system division.

The sequencing and synchronization software has been moved back into the background system, which also hosts the command interpreter METAI. Note that SEQUENCER now spawns the re-entrant task BOPTSK, which was access to the buffer-formatting software associated with METAI. BOPTSK is created at need by the sequencing software as before, and causes the foreground system to create a corresponding FOPTSK. It is FOPTSK that actually calls the operational level software which performs the command function. Strictly speaking, BOPTSK is no longer a shroud. It is rather an interface between the sequence control software and the operational routines. Again, any number of BOPTSK/FOPTSK pairs may exist at any given time; both tasks are fully re-entrant.

The differences between the systems represented by Figure 3 and Figure 4 are striking. The astute reader will note that the number of ground-to-ground transactions required to execute a given command has increased from one to five. While this cannot be described as desirable, it was nonetheless sufficient to meet our program requirements, and meet them on cost and schedule.

Clearly, were it not for the ability to establish logical, rather than physical, data and control connectives this task would have been much more costly to accomplish. Also, the ability to separate

architectural level functions from applications-specific functions was critical to the success of this endeavor.

A further point should be made to illustrate the power of these tools. The period of time between the development of the design for the new skeleton and its delivery to applications programmers as the operational environment of their software was approximately two weeks. The first week was consumed in the writing of the tasks to support the foreground system and their installation in the system. This amounts to about one thousand locations of code, distributed among five tasks. This does not include rather minor modifications to existing software. The debug effort took another week, and most of that effort was performed by a programmer who was not involved in the design or implementation of the new skeleton until that point.

This is not a trivial point. It shows that a programmer can step in and debug a major and completely unfamiliar piece of software in a very short period of time. This is due to the readability offered by the code level macros and the structure attainable with the higher order software tools. Programmers no longer have the need to debug the mechanics of inter-task communications, task spawning and ground-to-ground communications. They can now concentrate exclusively on the user code, and better comprehend the role of that code in the system.

One final point before proceeding into a quantitative discussion of the benefits we have realized through the use of these software tools. It is important when constructing large systems to get the skeleton system up quickly. Only in this way can applications programmers be applied with the operational environment in which their software is to reside. The first pass at debug and integration can be done in situ

rather than in some hastily contrived test frame which, after all, must be considered throw-away code.

QUANTITATIVE DISCUSSIONS

One must ask, of course, how much of the benefits we perceive are due to the fact that much code has been transported between these systems. One might counter with the argument that software tools are in part responsible for that capability, but that argument is not sufficient to make our case. Figure 5 illustrates the growth in code space of these systems, and the amount of new code generated to fulfill new functional requirements. Each of these systems was significantly larger than the previous system, and it might be added that each system was progressively more complex both in terms of software architecture and functional requirements.

Figure 6 is a graph of the calendar time allotted to each project. Crew size variations between these projects were small. Note that each system was allotted a progressively smaller period of time for design and implementation. Thus, with each project the team was creating progressively larger and more complicated systems in less calendar time.

Figure 7 is a graph of the labor required to establish the skeleton in each system. Additional information, such as ROLM experience, code space sizes and new code requirements, crew allocated to skeleton implementation and design, and the amount of calendar time required to establish the skeleton are presented. The labor cost to skeleton diminished with each successive project. It should be noted that each project had access to a successively higher level of software tools, most of which had been developed in the previous project.

System 1 programmers had access to only the code level macros, and only after about fifty percent of the labor had been expended. The first logical constructs were a by-product of that effort, and were highly valued by the personnel working on that task. General systems utilities were constructed and the code level macros improved in the interim period between System 1 and System 2, with most of the real work being done in the development of that system. While one would expect this to drive up labor cost to construct the skeleton, we see that labor cost actually dropped from 55 to 16 man-months. It should be pointed out that the skeleton of System 2 is much more complicated than that of System 1.

The last two levels of the software hierarchy were developed during System 3's design and implementation. Much of the effort went into tools designed to facilitate ground-to-ground communications, especially the ROUTER facility mentioned earlier. Towards the end of that project, the general multi-tasking structures became available, but due to their lateness were not fully used until System 4 development. Despite the additional complexities of ground-to-ground communications and a totally new set of functional requirements, System 3 skeleton labor costs dropped to 9 man-months.

Note that crew sizes on the skeleton system effort have also dropped steadily, meaning that a greater percentage of the team was available immediately for the design and development of applications-specific software.

This is dramatically reflected in Figure 8. The percentage of labor cost expended on skeleton system implementation and design dropped from 50% in System 1 to 5% in System 4. This means that 95% of the labor budget was available for the development of applications-specific code,

debug, and system integration. In systems of this size and complexity, with large user I/O requirements, this integration is, of course, a time of hectic activity and great anxiety. It is very desirable to have at hand the labor and time resources necessary to tackle this phase of system development in great abundance.

The software tools are of assistance in this aspect of systems development. Because the skeleton is up and running quite early in the project, applications-specific software can be debugged in a good approximation of the final operational software environment. Applications-specific software can be added to the skeleton as it is developed, resulting in the early detection of most integration problems. In system 4, most integration problems were solved prior to the calendar date set for the start of system integration.

As a result, our integration and validation processes were completed with few problems, most of them very minor. Field testing revealed a few minor problems which were patchable on site.

We have shown in our case example that one task out of more than twenty permanent system tasks caused a major revision of the original architectural level design of the system. It was one of the few tasks in which programmers did not make extensive use of software tools as they became available. It violated our overall design philosophy for separation of applications-specific code from architecture support code, and used physical rather than logical data and control connectives to communicate with supporting modules which served two purposes.

It is not remarkable that the original job estimator and system designer did not fully account for the problems of interfacing with this module. It is a massive piece of software which fulfills a complicated

functional requirement. Rather, it is remarkable that the alternative of changing the architecture to work around the problem was more cost-effective than performing massive surgery on the offending module.

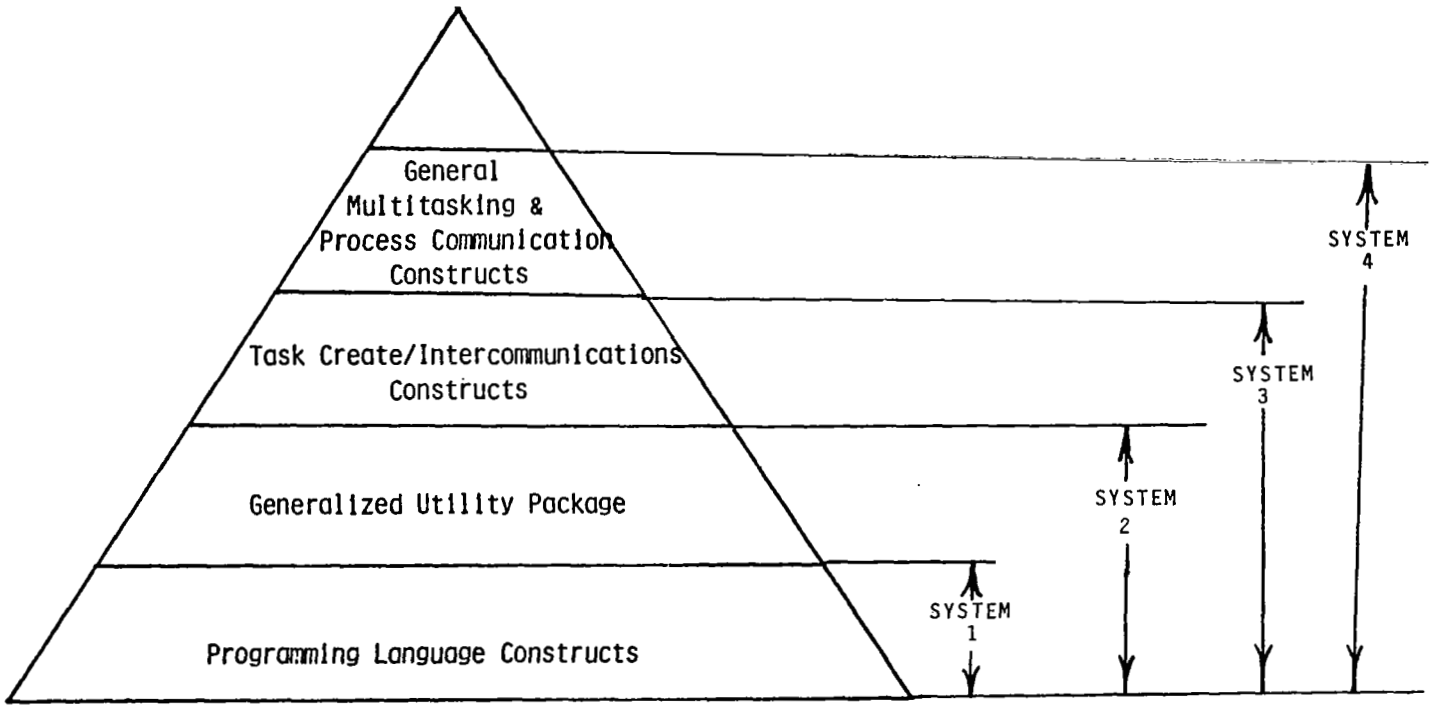
We conclude two things from this experience. First, that the use of the software tool set must be uniform throughout the system development team in order to ensure a high degree of transportability of software components. Second, without the software tool set, the job estimator would have been forced to concede to the customer that the job was grossly under-bid. We would not have made our cost and schedule.

Our quantitative discussion shows that there is a strong basis for saying that the software tool set dramatically impacts certain job performance criteria. Labor costs for almost every phase of systems development have dropped with each successive layer of the software tools hierarchy.

Given these conclusions, one may state with some firmness that the software tool set was fully worth the cost of developing and maintaining it. The software tool set permits this team to perform more complicated tasks in shorter periods of time at lower cost. As this greatly increases our ability to compete with other firms in our market area, the benefits of the software tools must be said to justify the expenditures to develop them.

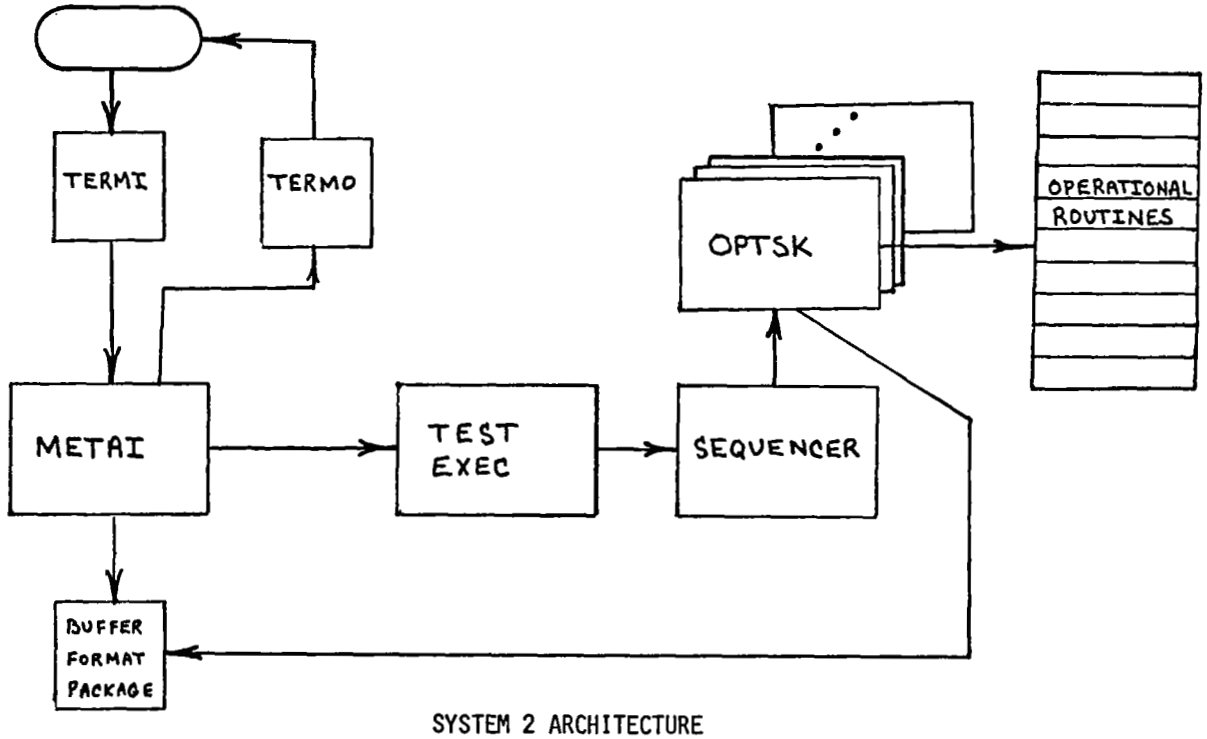
REFERENCE

1. Christofferson, M. J.: Real Time Software Tools and Methodologies. Ruggedized Minicomputer Hardware and Software Topics - 1981, NASA CP-2206, 1981. (Paper no. 8 of this compilation.)



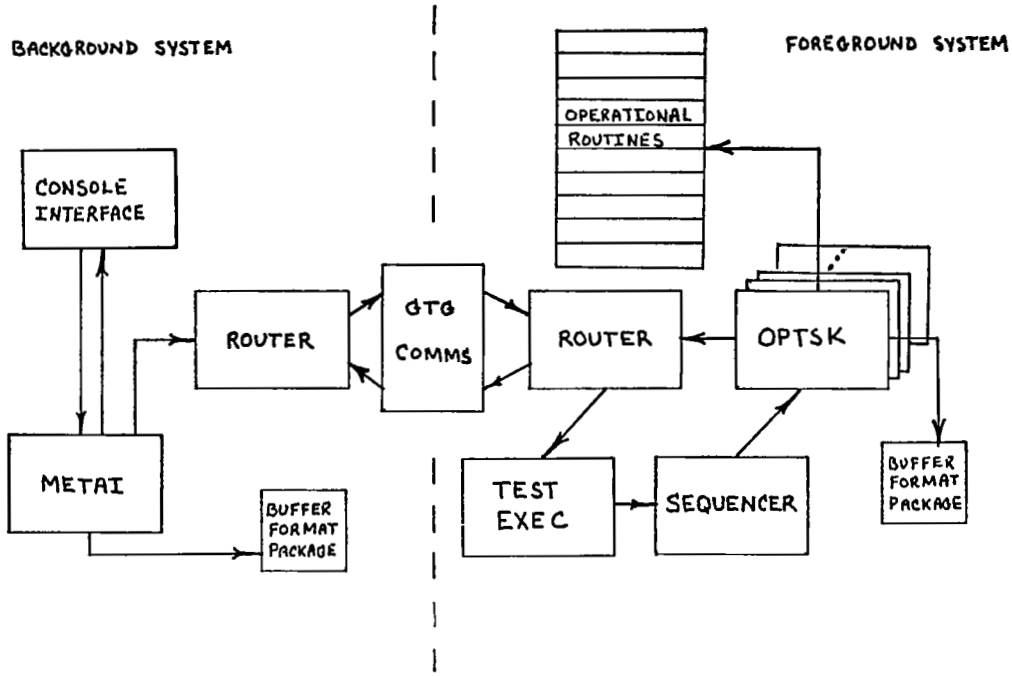
AVAILABILITY OF SOFTWARE TOOLS

Figure 1



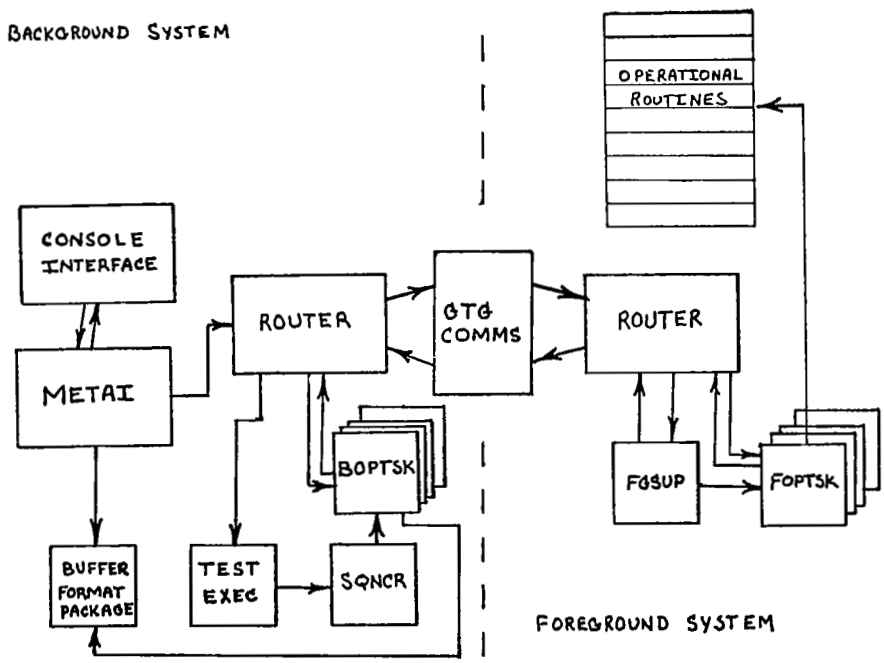
SYSTEM 2 ARCHITECTURE

Figure 2



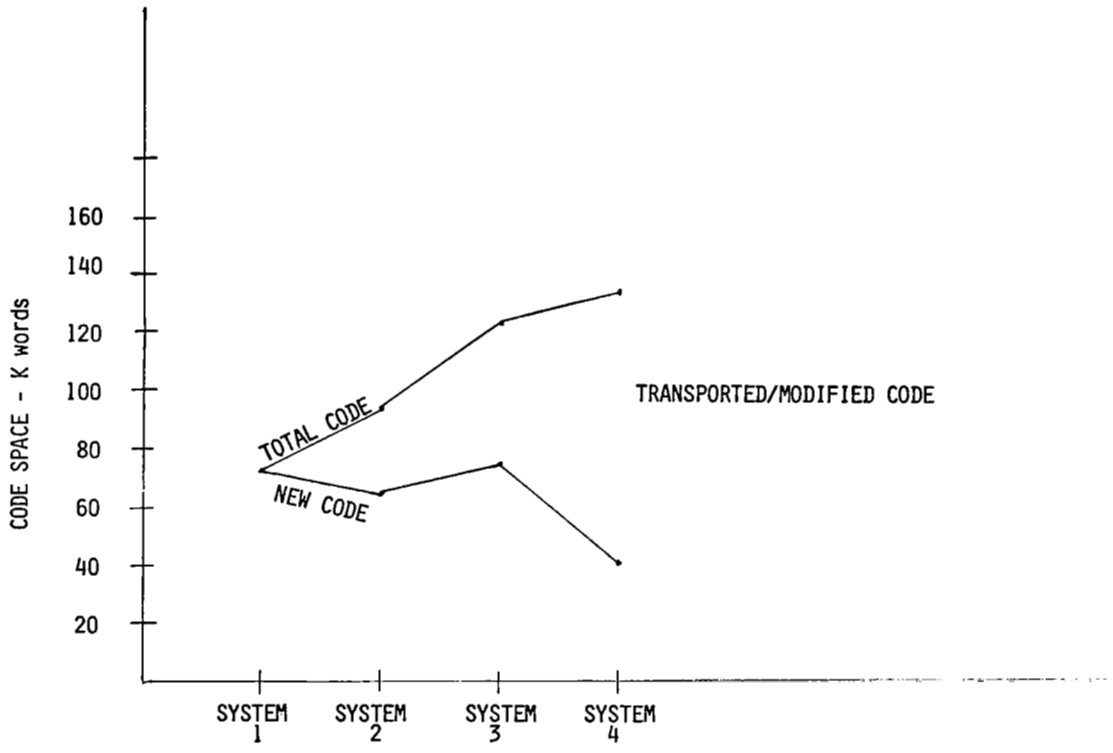
ORIGINAL SYSTEM 4 ARCHITECTURE

Figure 3



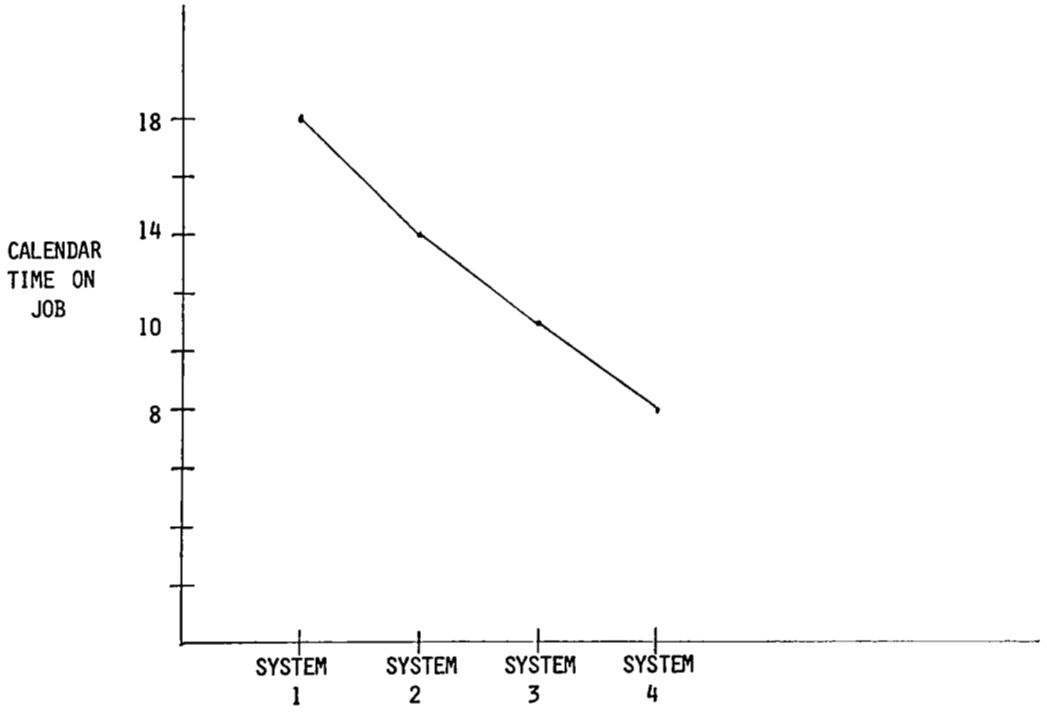
FINAL SYSTEM 4 ARCHITECTURE

Figure 4



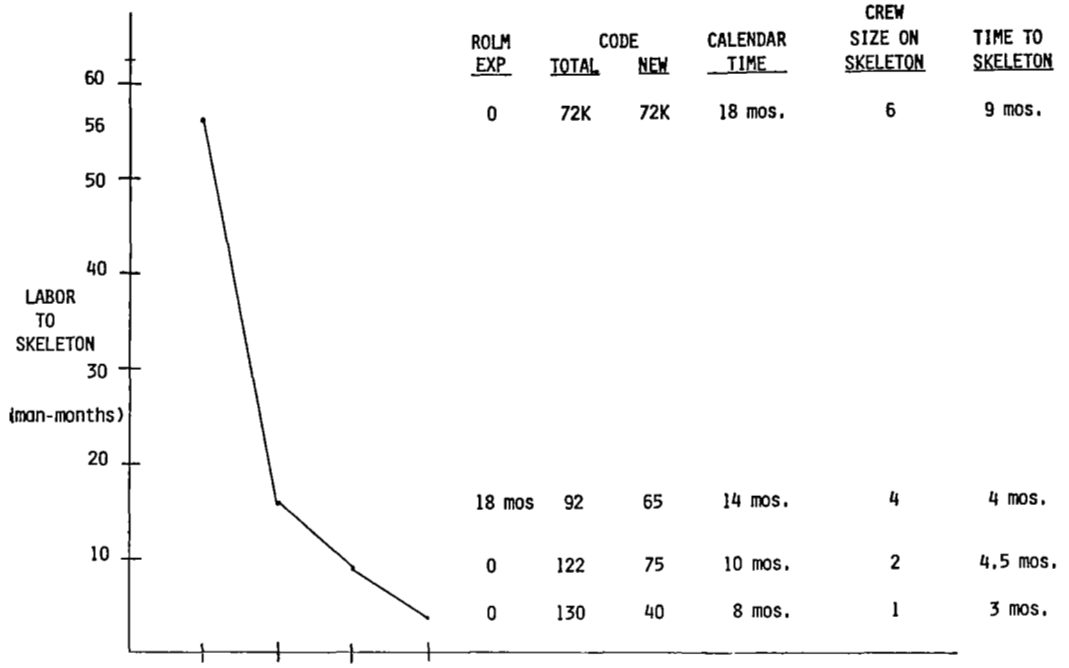
TOTAL AND NEW CODE SPACE

Figure 5



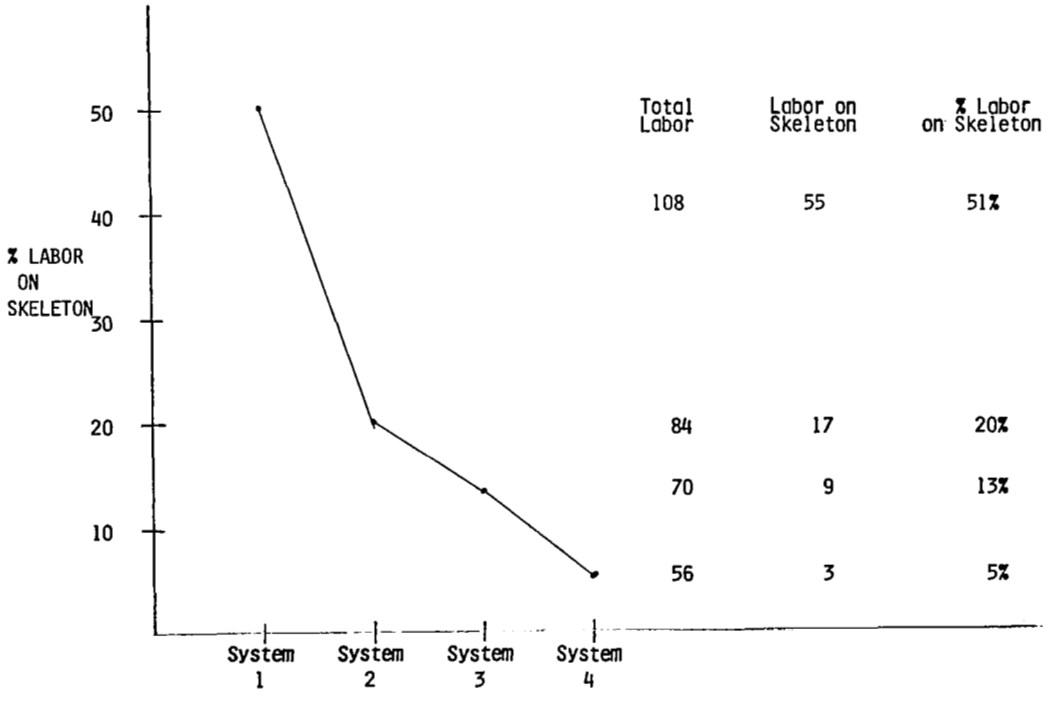
JOB SCHEDULE CALENDAR COMPRESSION

Figure 6



LABOR COST TO ACHIEVE SKELETON

Figure 7



% Labor Expenditure on Skeleton

Figure 8

ROLM COMPUTERS IN THE FLIGHT TESTING OF THE FOKKER F29 AIRCRAFT

P. J. Manders
National Aerospace Laboratory
Amsterdam, The Netherlands

ABSTRACT

Since 1919, the National Aerospace Laboratory has been the central institute in The Netherlands for research performed for civil and military aeronautics and spaceflight technology. The Netherlands Aircraft Factories, Fokker, is developing the Fokker F29 Aircraft (short-haul, twin-jet, second generation, high-bypass-ratio engines; supercritical wing; advanced avionics (ARINC 700); autoland cat. III; 138-156 passengers). Flight tests will take place simultaneously in 1984 with three prototypes. For the evaluation and certification flight trials with the F29 prototypes, NLR, in close cooperation with Fokker, is responsible for the design, development, installation, and operation of the test equipment (MRVS) under contract with NIVR, the Netherlands Aerospace Agency.

The main requirement of the MRVS is to continuously record on an instrumentation recorder data of up to 1500 parameters with a total sample rate of up to 10 000 samples per second. After the flight the tapes will be processed on the NLR-Fokker computer network.

In order to compress the evaluation and calibration time period, the following additional requirements were set for two test systems:

(a) Recording of selected parameters, time-tagged on computer-compatible tape (CCT)

(b) Recording of selected high-bandwidth signals and ad hoc parameters on analog tape

(c) On-board presentation of calibrated parameter data, in engineering units, in numerical as well as graphical form for:
- System check-out during pre-, in- and post-flight
- Quick-look analysis during in-flight

(d) Real-time presentation on the ground by telemetry for:
- Flight monitoring
- Take-off and landing measurements
- Noise measurements

(e) Limited data processing on board

(f) Data processing on the ground on the Fokker-NLR computer network
- Constant processing of the CCT and the analog tape, with selected data
- Occasional processing of the instrumentation tape with all parameters recorded during the entire flight

To meet these requirements a computerized F29 Flight Test System was designed, the F29 Measurement, Recording, and Processing System (MRVS). This paper discusses the development and characteristics of the on-board computer system (OBC) as a subsystem of the MRVS.

INTRODUCTION

The National Aerospace Laboratory (NLR) is the central institute in The Netherlands for research performed for both civil and military aeronautics and space flight technology. The laboratory carries out theoretical and experimental research in support of development projects of the aircraft industry. The NLR is a non-profit foundation that obtains its financial resources mainly from work under contract and from the government.

Fokker is developing a new aircraft, the Fokker F29. The NLR supports Fokker in studies on wing technologies, airframe integration, engine noise reduction, wind tunnel experiments, and in the evaluation of modern materials (fig. 1). Under NLR responsibility, Fokker and NLR together are developing and realizing the F29 flight test system.

FOKKER F29 AIRCRAFT

The Fokker F29 aircraft is a new generation of short-haul, twin-jet aircraft for about 160 passengers, with second generation high-bypass-ratio engines, yielding low noise production. The aircraft has supercritical wings, advanced avionics systems including ARINC 700, and category III autoland equipment. Two of the three prototypes that will be built contain a complete flight test system. The third one contains a limited system.

Figure 2 is a diagram of the F29 evaluation and certification flight schedule. In this diagram, the x axis shows the calendar time and the y axis shows the block hours spent in testing.

The first flight of prototype I is scheduled for the spring of 1984; the first flight of prototype III will be six months later. The test flights of prototypes I and II are almost simultaneous and will be completed in somewhat more than one year (fig. 2).

FLIGHT TESTING, FUNCTIONS HISTORY, AND TRENDS

The primary functions of flight testing (fig. 3) are:

(1) Measurement of the aircraft parameters in which the designers and flight test engineers are interested

(2) Recording of the measured parameters. Recording is necessary because complete in-flight and real-time interpretation and processing of the measured data is impossible. Recorded data will also be used for statistics.

(3) On-board presentation of the measured data, in engineering units, for quick-look analysis. This gives the flight-test engineer the option of deciding during the flight whether to continue, change, or interrupt the test flight.

(4) Data processing of the recorded raw data

(5) Analysis of the processed data

The trends in flight testing are to measure, record, and present more data, i.e. more parameters at higher sample rates. On the other hand, there is a trend toward shorter turn-around times in data processing and analysis. The larger number of transducers and the shorter turn-around times required necessarily lead to highly automated flight test systems.

In 1937 the NLR developed a so-called Automatic Observer, a combination of a remote-controlled camera and an instrument panel for the intended flight test (fig. 4a). The camera method provided on-board presentation and recording functions. Parameters were presented in engineering units on the built-in instruments while all the parameters were recorded on film. One to several pictures per second were taken. The film was available for interpretation within several hours after the flight. In the early years the interpretation was done manually, while in later years digitizing instruments were used. At that time processing and analysis were done manually and with the help of simple mechanical and electromechanical calculators.

Figure 4b shows that nowadays, apart from a large number of parameters, the data of modern avionics systems also have to be measured. The data acquisition system multiplexes and digitizes the data and records the data on a magnetic tape. The on-board computer enables presentation of the data on visual display units in each desired format. On-board computers also enable data processing during and shortly after test runs. After the flight, the data processing and analysis take place on ground computer systems.

Figure 5 shows the history of the NLR-Fokker flight test systems from 1950 to the present, including the characteristics of the F27 Friendship, the F28 Fellowship, and the aircraft under development, the Fokker F29. The number of parameters has increased from 70 to 1500 and the sample rates from 40 to 10 000 samples per second. The type of recording systems and the method of data processing have also changed. In 1967 a ground computer was introduced, while nowadays airborne computers are used. For the first time a telemetry link was also introduced.

F29 FLIGHT TEST SYSTEM

The F29 Flight Test System is called the Meet- Registratie- en Verwerkings Systeem, MRVS, which means measurement, recording, and processing system. The tasks to be performed by the MRVS are listed in figure 6. The main constituents of flight testing are recording, presentation, and processing.

The requirements for recording are as follows:

(a) During the entire flight, data of all the measured and digitized parameters have to be recorded continuously on an instrumentation recorder

(b) During the various test runs of a flight, a subset of the measured parameters has to be selected, time-tagged and recorded on a computer-compatible tape

(c) Selected high-bandwidth signals and ad hoc parameters will be recorded on an analog tape recorder

Presentation on board is required for several purposes, such as:

- The pre- and post-flight system check-out
- Monitoring of the digital data acquisition and recording systems during test runs
- In-flight quick-look analysis

Real-time presentation on the ground is required for:

- Flight monitoring purposes
- Take-off and landing measurements
- Fly-over noise measurements

This real-time presentation on the ground is made possible by a telemetry link.

The third item, processing, requires limited data processing on board. For instance, during the test runs a limit check on all parameters will be done and results of these limit checks will be printed on a line printer. Between the test runs it will be possible to read the recorded data from the computer-compatible tape back into the on-board computer. The data can be processed using the in-flight program, which is a simulated repetition of the test run, or the data can be processed by special post-flight programs.

After the flight the computer-compatible tapes and the analog tapes will be processed on the Fokker-NLR computer network. In case of an assembly failure or in case of an incorrectly prepared test or equipment configuration with the effect that the computer-compatible tapes do not contain the desired data, the instrumentation tape will be processed on the ground-based computer network, at the cost of a much longer turn-around time.

Figure 7 shows a simplified block diagram of the F29 Measurement, Recording, and Processing System (MRVS). The complete MRVS consists of 14 on-board subsystems and 24 subsystems on the ground. The upper half of figure 7 shows the collection of on-board systems with inputs coming from the avionics systems and the transducers. Apart from these, the on-board system receives a mag tape δ, with the parameter selections and calibration data both necessary for each particular test run. The On-Board Computer System delivers, as an output, the computer-compatible tape, β, for recording only during a test run. Other outputs of the on-board systems are the digital tape, α, for recording during the entire flight, the analog tape, ν, and the telemetry link.

The lower part of figure 7 shows the MRVS subsystems on the ground, such as the development and processing systems of NLR and Fokker, the Fokker Central Computer, and the computerized Telemetry Ground Station. These subsystems deliver the data required for analysis.

Figure 8 shows more detail of the on-board systems. Three data acquisition systems can be distinguished:

(a) The Avionics Data Acquisition System, which delivers five digital serial data streams to the continuous recorder

(b) The Digital Data Acquisition System, which generates eight digital serial data streams to the same recorder

(c) The Analog Data Acquisition System for ad hoc parameters and high-bandwidth signals, with its own analog tape recorder

Besides the five avionics channels and the eight digital channels, the continuous recorder receives one channel from a Time Code Generator. These time data provide synchronization and off-line search possibility for the recorded data. All the data channels and the time data are also fed into the On-Board Computer System.

ON-BOARD COMPUTER SYSTEM

The On-Board Computer System, OBC (fig. 9), is mainly brought into the F29 Flight Test System to expedite a number of processes:

(a) The flight preparation phase can be shortened by partly automating the pre-flight check-out of the on-board data acquisition and recording systems.

(b) The qualification of success or failure of a test run in an early stage is made possible by means of the computer and its visual display units, on which selected and calibrated parameter data will be presented.

(c) During the data processing on the ground after the test runs, a time-consuming tape-conversion action can be omitted. The OBC offers the possibility of recording a selection of all the sampled parameters in a standardized computer-compatible format. Usually this CCT tape (β) will be processed instead of the so called continuous tape (α).

Besides the speeding-up possibilities, the OBC will be used for monitoring, checking, and controlling of the on-board data acquisition systems.

OBC Tasks

Figure 10 presents the OBC tasks as follows:

(a) Recording of selected, non-calibrated, time-tagged parameters

(b) Presentation of selected calibrated parameter data in engineering units, in numerical as well as in graphical form

(c) Limited real-time processing

The on-board computer functions are presented in figure 11; the two data highways are easily recognized. As described before, the data are delivered by thirteen mutually unsynchronized data acquisition assemblies in a hardware programmable frame with a speed of about 500 to 2000 data words per second. The total

number of parameters was not supposed to exceed 1500 but, as usual with systems under development, user requirements increased and at the moment stand at 1800 parameters. The total sample rate at the input side is limited to about 10 000 samples per second. Another indispensable input to the OBC is the Time Code Generator which provides the time each millisecond.

Two important functions in the OBC are parameter selection and data calibration, which are test-run and configuration dependent. The information for selection and calibration will be delivered on the mag tape δ, which will be generated on the NLR ground computer network. During the test run these data are continuously and immediately accessible. The first action to be executed on the incoming data is to identify the data words, which means obtaining the specific data acquisition unit as well as the specific data word in the frame. The two characteristics have to be added to the data word, a process called labeling. From there on two processes are distinguished, namely recording on computer-compatible tape (via the right-hand data highway) and data presentation (via the left-hand data highway).

The presentation process contains more activities than simply the presentation of parameters on the two display units for operator and observer (figs. 12 and 13). Included in this process are:

- Analog outputs to a trace recorder and to analog displays
- Digital outputs to a Central Warning System and to digital displays
- ARINC output, among which are the tuning data for the DME Interrogator
- IEEE commands to a plotter, only for off-line purposes
- Messages and debriefing data to the line printer

A comparison of the two processes demonstrated that for the recording process, every selected data word has to be time-tagged with the time given by the Time Code Generator, and that data and time have to be recorded together. For the presentation process it is necessary for the most recent data word to be available for each parameter at every moment. On request these data words will be read and used in further activities.

After making a budget of the number of parameters to be delivered by these two processes, it was found that in the recording process the worst case selection was about 500 parameters, with a total sample rate of about 3200 samples per second. For the presentation process one can count on a limitation of about 100 parameters, with a total sample rate of 100 samples per second. The functions to be executed by the recording process are, in principle, rather simple. Every identified and labeled data word has to be checked to determine whether it belongs to the CCT selection. If it does not, the data word will be ignored. If it does belong, the time of the Time Code Generator will read, and the label, the data word, and the related time word have to be transferred to the CCT tape β.

The functions to be executed during the presentation process are presented in the left-hand data highway of figure 11. In principle all the identified and labeled data words have to be stored in a memory. Each parameter of each data acquisition unit will have its own address in this memory, so that each time a new data word from the same transducer over-writes the memory position of this parameter. In this way a Most Recent Parameter Buffer, the MRPB, is created. The printed circuit board to implement this buffer is shown in figure 14. Since the added labels are unique for each parameter, the addresses of the MRPB can be

derived from the labels. After every update the time will also be written into the MRPB. During execution of a test run the data of the Most Recent Parameter Buffer will be read every two seconds and calibrated into engineering units, and will be presented on the two display units.

OBC Design

In all, eight configurations were examined and evaluated with respect to cost, expected development time, modularity, complexity, and throughput rate or computer CPU load. Several of the configurations were derived from our first generation data acquisition system with a ROLM 1601 computer. Some configurations with two computers were considered, but apart from the cost aspect an extra effort was expected in the required computer-to-computer data transmission. Figures 15, 16, and 17 show three of the examined configurations.

Figure 15 shows the first configuration, in which the functions, as far as possible, are realized in software. Every data word of the thirteen data acquisition units causes an interrupt request to the computer, where each data word is identified and labeled. The complete recording and presentation process is done in software. A calculation demonstrated that counting with 10 000 interrupts per second would require a CPU load of more than 100%. This solution was therefore useless. Realization of the identification and labeling actions, as well as the CCT selection in hardware, would be an improvement, but the interrupt actions would still result in an unsuitable duty cycle of about 70%.

In figure 16 a configuration is shown in which the recording and presentation processes have been separated from beginning to end. For both processes, as much as possible was realized in hardware. Realization of the CCT selection storage was done by introducing a RAM, which before the test run would be filled with the CCT selection data. It was necessary to create a hardware CCT data buffer, and data transfer from the CCT buffer and the MRPB buffer was done by direct memory access. Both buffers would be copied into the computer memory and would be accessible for CCT recording and data presentation. The CCT selection is fixed during a test run, while VDU selections are changeable. The expected time needed for input as well as output actions will take about 15% of the CPU load. Since direct memory access was used, this was seen as a minimum estimate. As some parts of the hardware are duplicated, this configuration is not the best one with respect to cost, volume, and weight.

In phase III (fig. 17) the duplications in the hardware were removed. This could only be done by introducing an internal bus system with a bus controller. The software activities were the same as in phase II. After identification and labeling, every data word was transferred via the internal bus. This configuration is the solution which was ultimately chosen. The hardware was realized in an NLR-ROLM Interface Unit, the ROLIN, a full ATR assembly.

Figure 18 presents a block diagram of the On-Board Computer System. The upper half of this figure gives the functions incorporated in the ROLIN. Underneath the dashed line are the two ROLM Data Channel Controllers used for the CCT and MRPB data transfer, and also the software functions. The ROLIN is suitable for sixteen input interfaces with input and labeling circuits and various control circuits.

For the internal ROLIN bus system the idea of the ROLM I/O bus was copied, with sixteen data lines, six device code signals, data transfer signals, and the interrupt control signals. This was a great help in the design of the input interfaces and the bus controller. Each data word transport is initiated by means of a hardware input request to the controller, which will only respond if the bus is free, first by reading the related device code and then by reading the relative label and data. The label is converted to an MRPB address by means of an EPROM, and the data word is written into the MRPB. The label will be checked simultaneously against the contents of the CCT selection RAM, and if a match is found the label, data, and time are transferred to the CCT buffer. This CCT buffer is realized as a double buffer. After reading 256 parameters the contents of the buffer will be transferred to the computer memory. In the meantime, the other half of the CCT buffer can be filled by new parameters. Under software control the MRPB and CCT buffer copies in the computer memory are accessible for presentation on VDU and other peripherals and for recording on CCT. The CCT selection data in the ROLIN-RAM can also be refreshed and renewed via direct memory access.

Figure 19 presents the On-Board Computer hardware configuration. The computer is a ROLM 1664 computer with 64 K memory. The ROLM I/O Box contains the various standardized peripheral interfaces. Two Interstate Plasma Display Units serve as operators' terminal and observers' display. The CCT unit is a Miltope Mag Tape Unit. The local OBC files for selection and calibration data and for storage of pre-, in- and post-flight programs are available on the Miltope Floppy Disk System, which consists of one master and three slave drivers. Connected to two data channel controllers is the ROLIN with the CCT, MRPB, and bus control circuits and the input interfaces. With the ROLIN Monitor Unit it is possible to present the data of the internal bus system on digital displays for maintenance purposes.

In figure 20 a diagram of the software is given. The software consists of five programs:

(a) The Conversion Program converts the selection and calibration tape into the local OBC file

(b) The Calibration Program will be used to calibrate the entire measurement channel from transducer to digitized data

(c) The Flight Preparation Program is required for the more or less automated system check-out just before the flight

(d) The In-Flight Program controls all the recording and presentation activities during the test run (fig. 21)

(e) The Post-Flight Programs contain various processing activities which will be executed after the test run

CONCLUSION

In this paper it is demonstrated that only with the help of on-board computers is it possible to meet the growing requirements in flight testing. These requirements are:

- More parameters to measure
- Higher sample rates
- Shorter turn-around times
- More effective on-board data presentation
- On-board processing

With a team of six men the On-Board Computer System is now being realized as follows:

- Such technologies as hybrid circuits and flexible print wiring are being incorporated into the ROLIN

- Concerning the software, work is proceeding on the technical design of the In-Flight Program and the Conversion Program.

In the course of this year the computer, the ROLIN, and the software will be integrated into a minimum system, and in 1982 the OBC system will be completed.

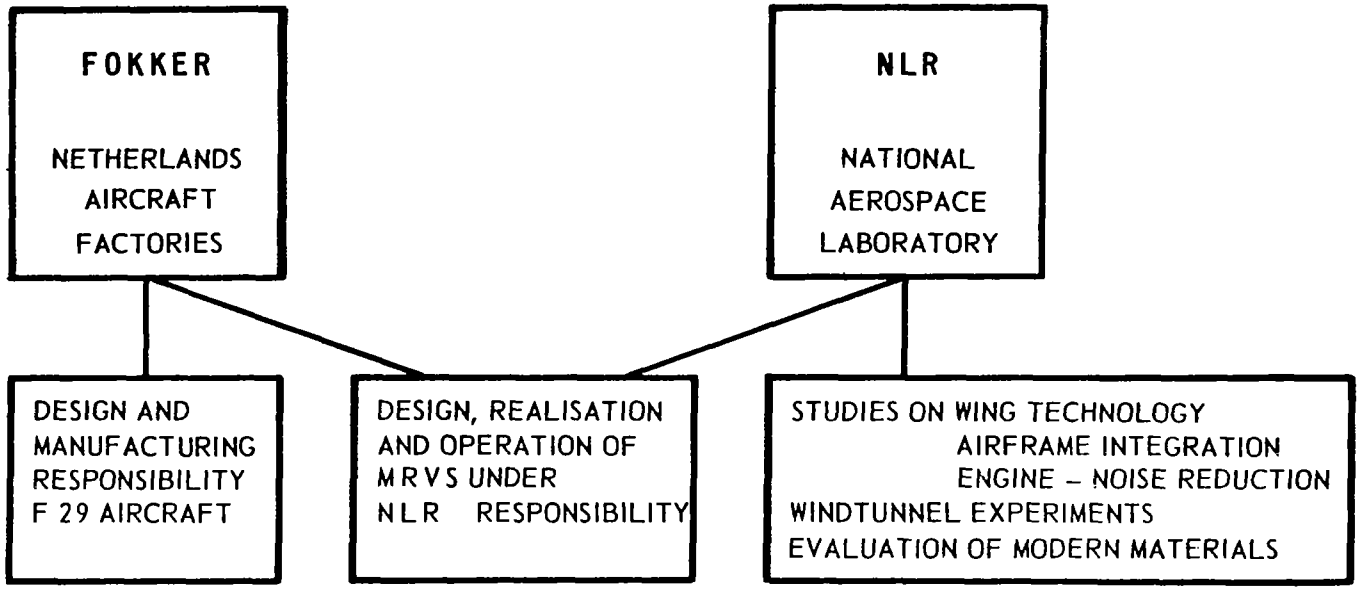


Figure 1

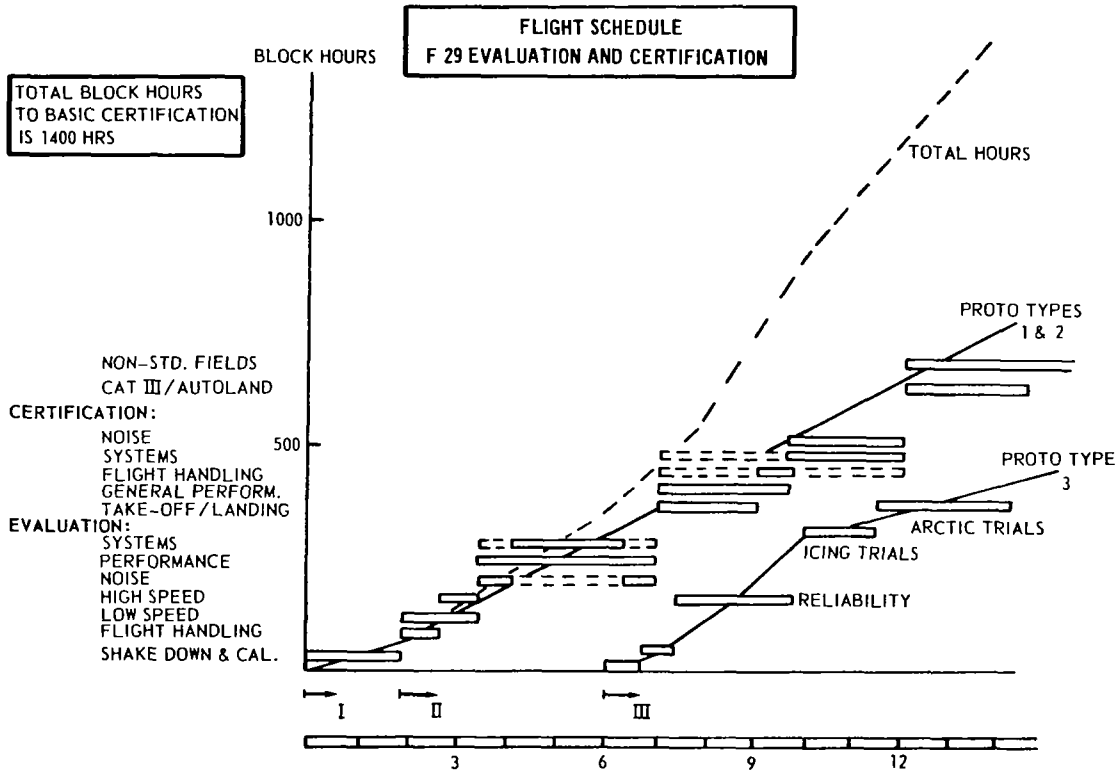


Figure 2

FLIGHT TESTING	
FUNCTIONS	TRENDS
MEASUREMENT	MORE DATA = MORE PARAMETERS, HIGHER SAMPLE RATES
RECORDING	
PRESENTATION	
DATA PROCESSING	SHORTER TURN AROUND TIMES
ANALYSIS	

Figure 3

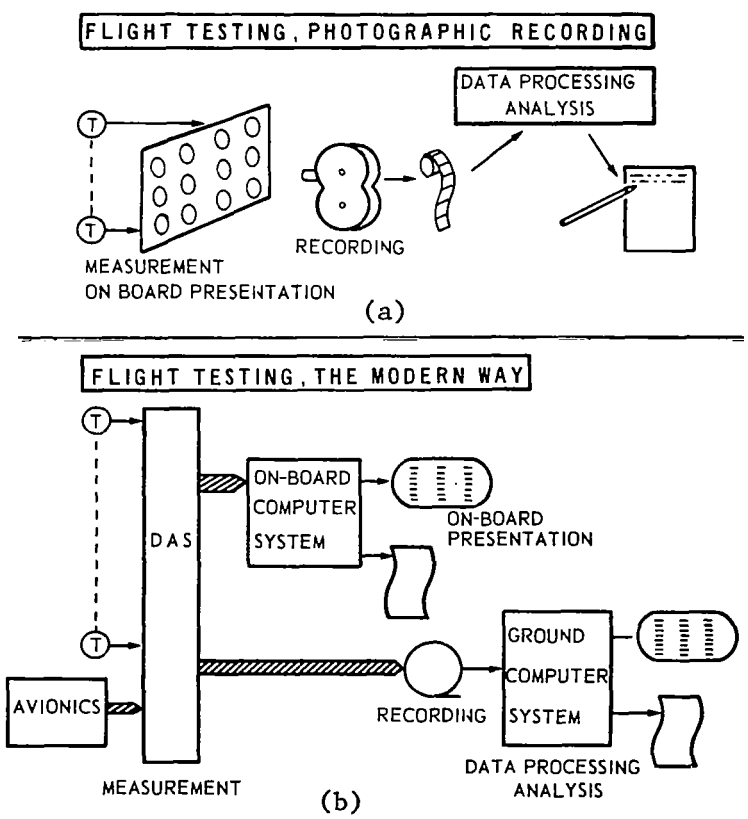


Figure 4

HISTORY OF NLR – FOKKER FLIGHT TEST SYSTEMS

AIRCRAFT	FRIENDSHIP	FELLOWSHIP		
	F 27	F 28		F 29
PASSENGERS	40 – 56	60 – 85		138 – 156
ENGINES	TWIN – TURBOPROP.	TWIN JET		TWIN JET
TAKE-OFF WGT	22 000 kg	30 000 kg		60 000 kg
FIRST FLIGHT	1955	1967		1984
MEASUREMENT				
NUMBERS OF PARAMETERS	70	50	110	1500
SAMPLES PER SECOND	40	50	550	10 000
RECORDING	FILM	MAG. TAPE	FILM	MAG. TAPE
PRESENTATION	INSTRUMENTS		INSTRUMENTS	COMPUTER VDU AND VIA TELEMETRY ON GROUND
DATA PROCESSING TURNAROUND TIME	MANUAL ONE DAY TILL SEVERAL DAYS	COMPUTERIZED 24 HOURS		COMPUTERIZED ≤ 24 HOURS

Figure 5

MRVS TASKS	
RECORDING	• ALL PARAMETERS (1500, 10.000 s/s), DIGITAL, ON INSTRUMENTATION RECORDER, DURING ENTIRE FLIGHT • SELECTED PARAMETERS, TIMETAGGED, ON COMPUTER COMPATIBLE TAPE (CCT) • SELECTED HIGH – BANDWIDTH SIGNALS AND AD HOC PARAMETERS ON ANALOG TAPE
PRESENTATION ON BOARD	CALIBRATED IN ENGINEERING UNITS, NUMERICAL AND GRAPHICAL , FOR • SYSTEM CHECK OUT PRE – IN – POST FLIGHT • QUICKLOOK ANALYSIS, IN FLIGHT
GROUND	BY TELEMETRY , FOR • TAKE OFF AND LANDING MEASUREMENTS • NOISE MEASUREMENTS
PROCESSING ON BOARD	LIMITED DATA PROCESSING
GROUND	ON FOKKER – NLR COMPUTER NETWORK • USUALLY CCT WITH SELECTED DATA AND ANALOG TAPE • OCCASIONALLY INSTRUMENTATION TAPE

Figure 6

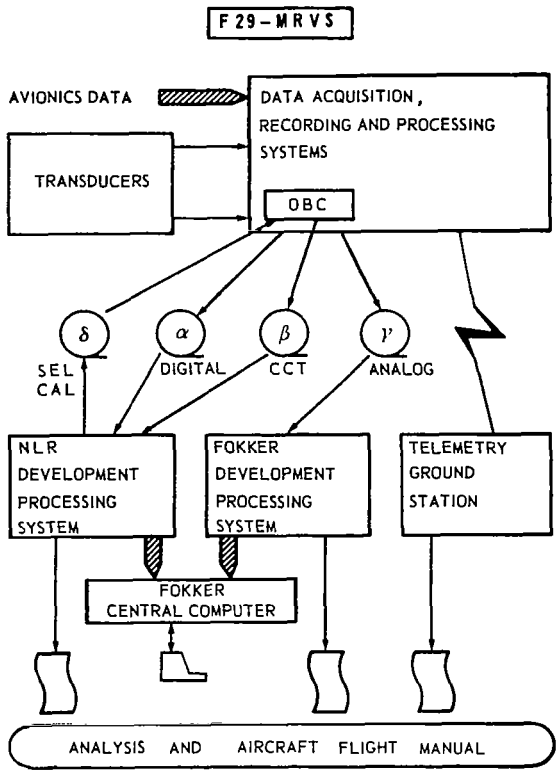


Figure 7

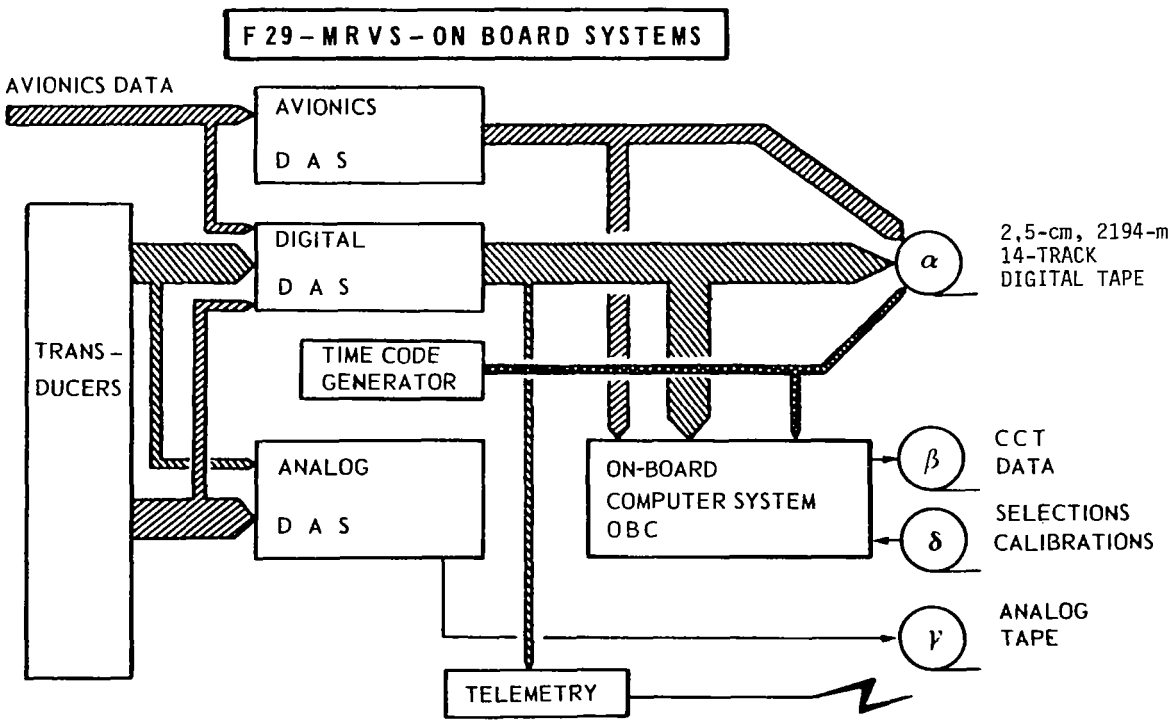
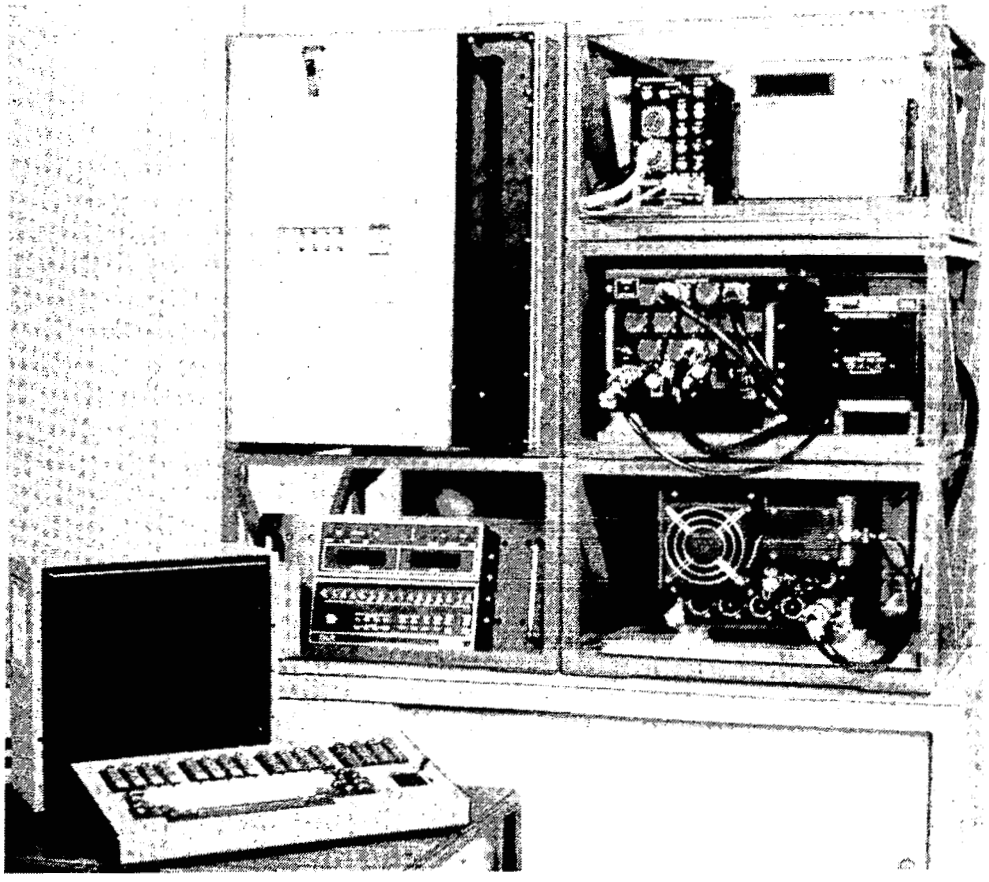


Figure 8



ON-BOARD COMPUTER SYSTEM

Figure 9

OBC TASKS	
RECORDING	SELECTED, NON-CALIBRATED, TIMETAGGED PARAMETERS
PRESENTATION	SELECTED PARAMETERS, CALIBRATED, IN ENGINEEREING UNITS, NUMERICAL AND GRAPHICAL, FOR : • SYSTEM CHECK-OUT, PRE-, IN-, POST-FLIGHT • QUICK-LOOK ANALYSIS, IN-, POST-FLIGHT
PROCESSING	LIMITED DATAPROCESSING IN-, POST-FLIGHT

Figure 10

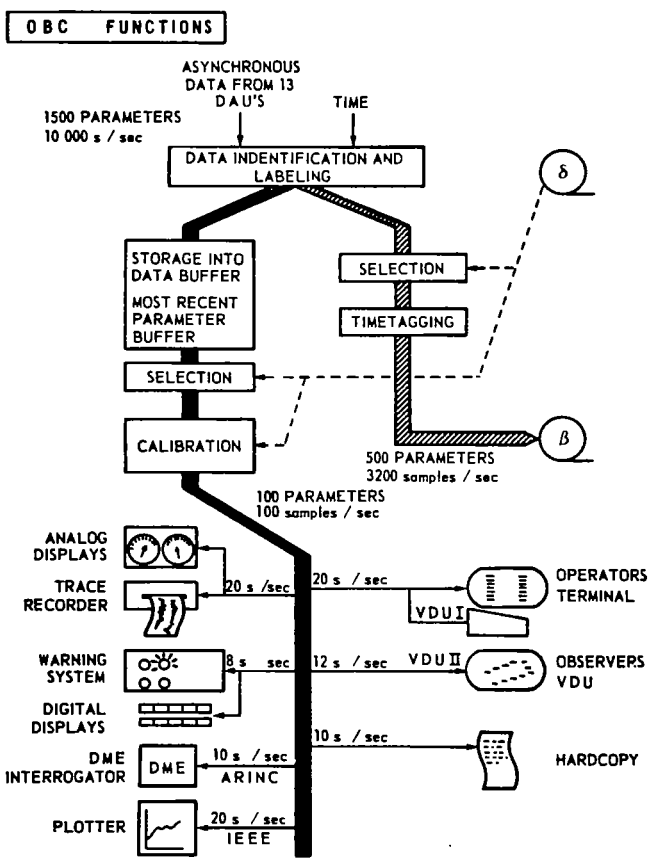
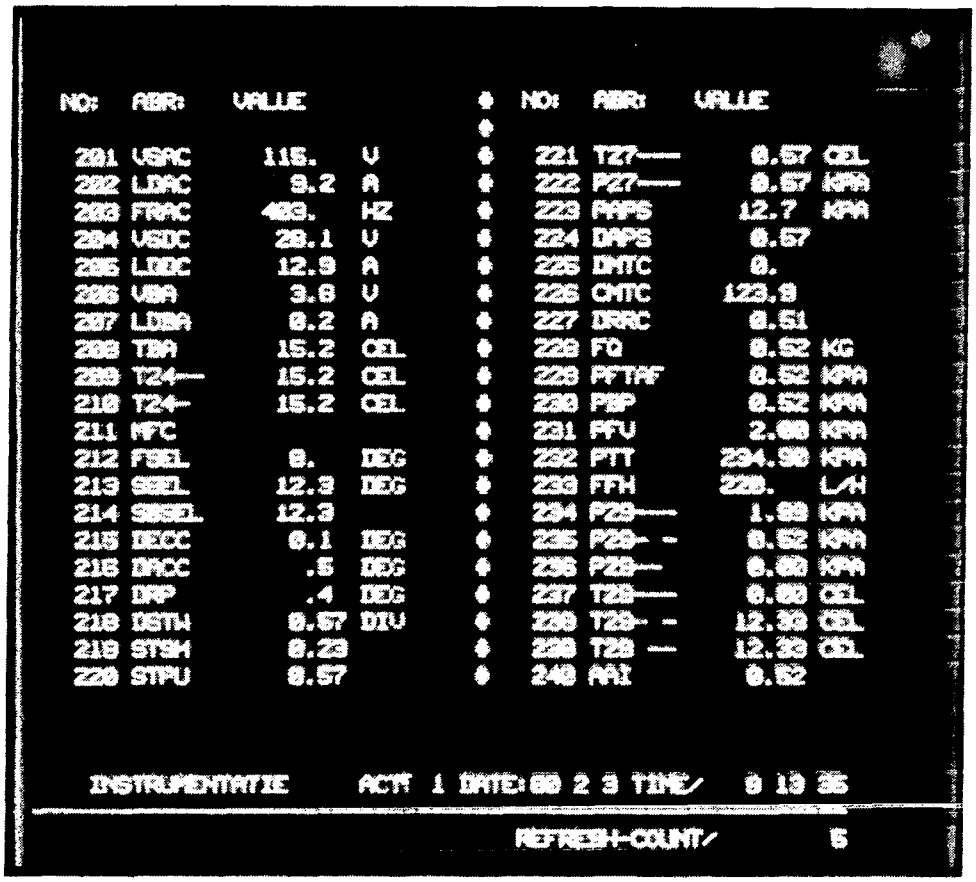
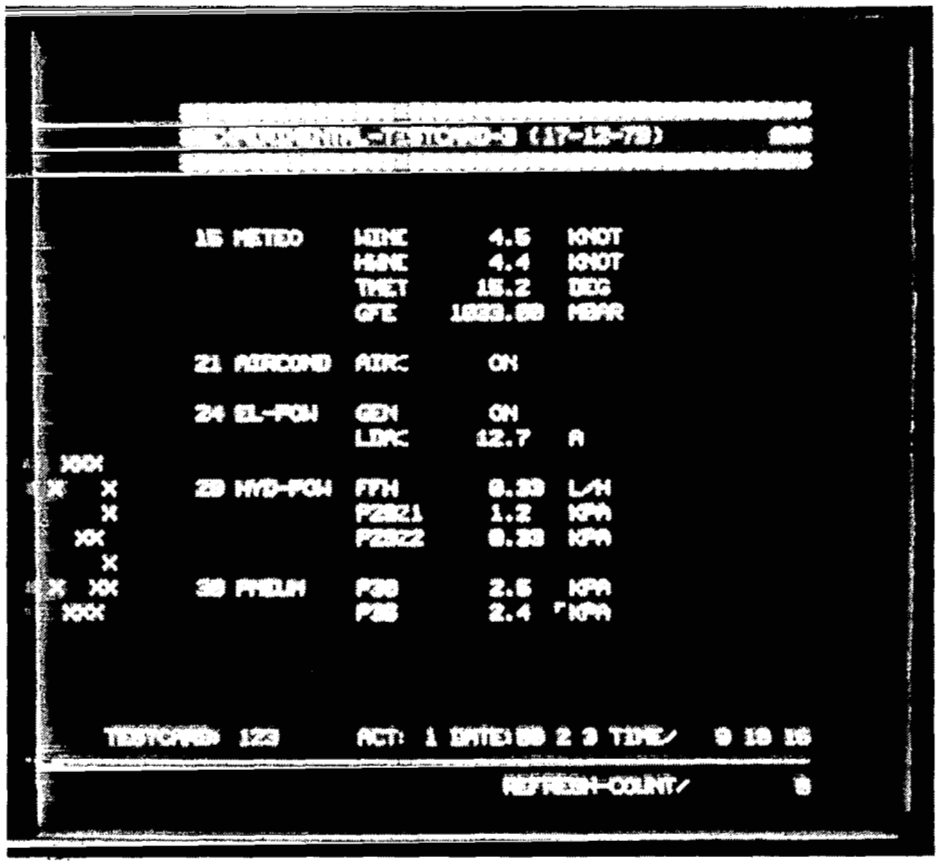


Figure 11



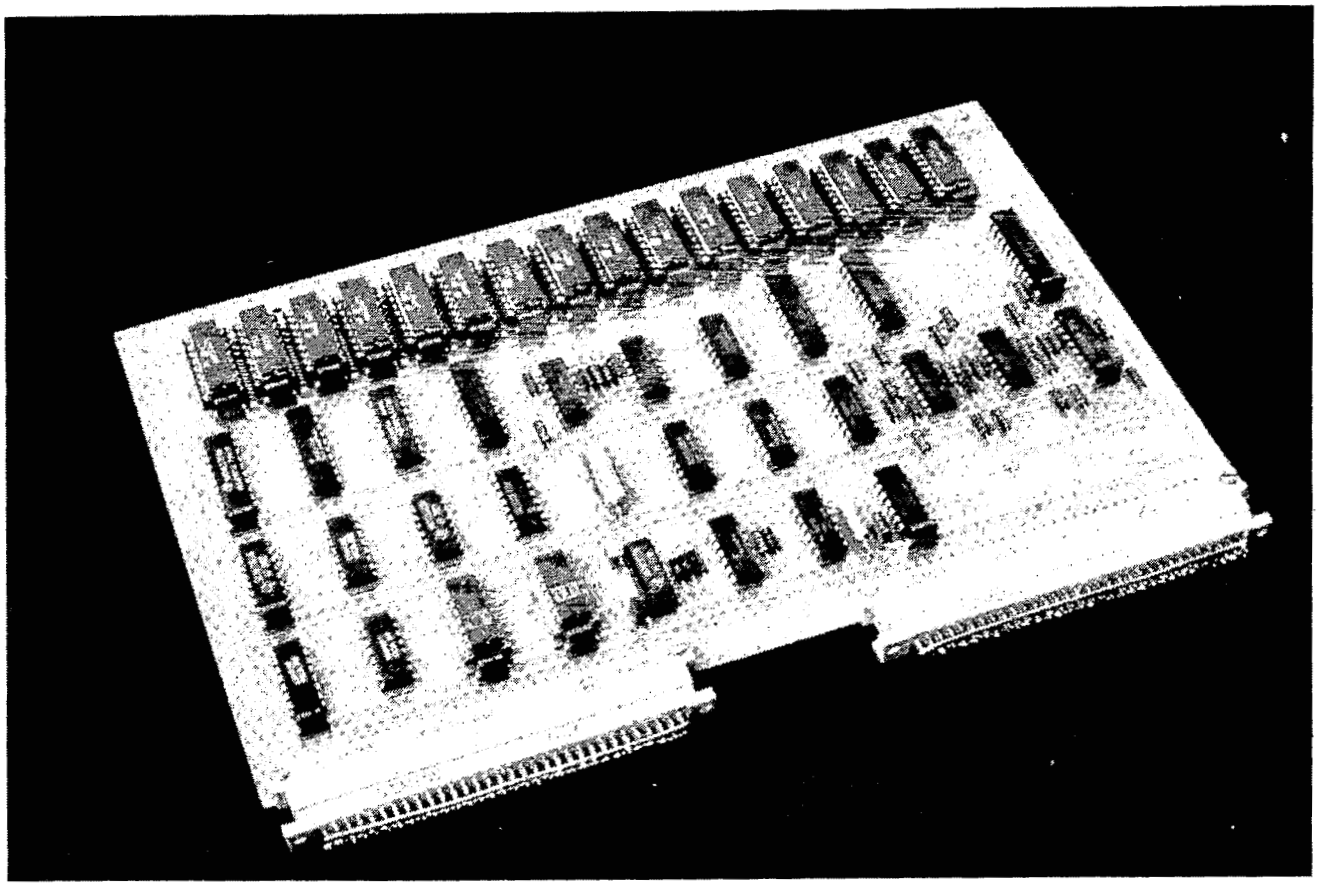
PLASMA DISPLAY
OPERATORS PAGE

Figure 12



PLASMA DISPLAY
OBSERVERS PAGE

Figure 13

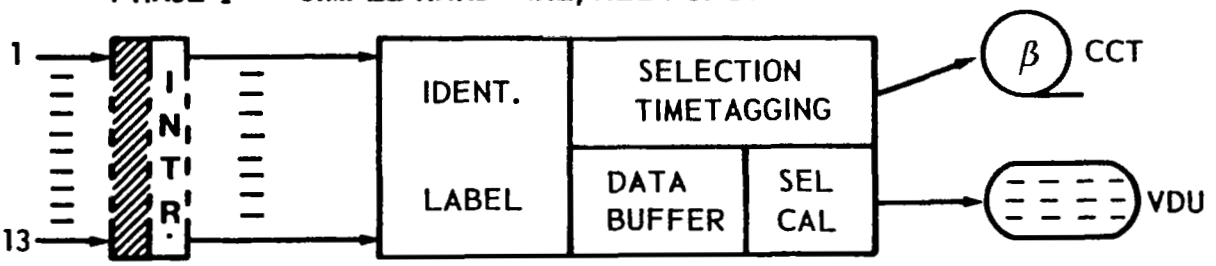


PCB MOST RECENT PARAMETER BUFFER

Figure 14

OBC DESIGN

PHASE I SIMPLE HARDWARE, ALL FUNCTIONS IN SOFTWARE

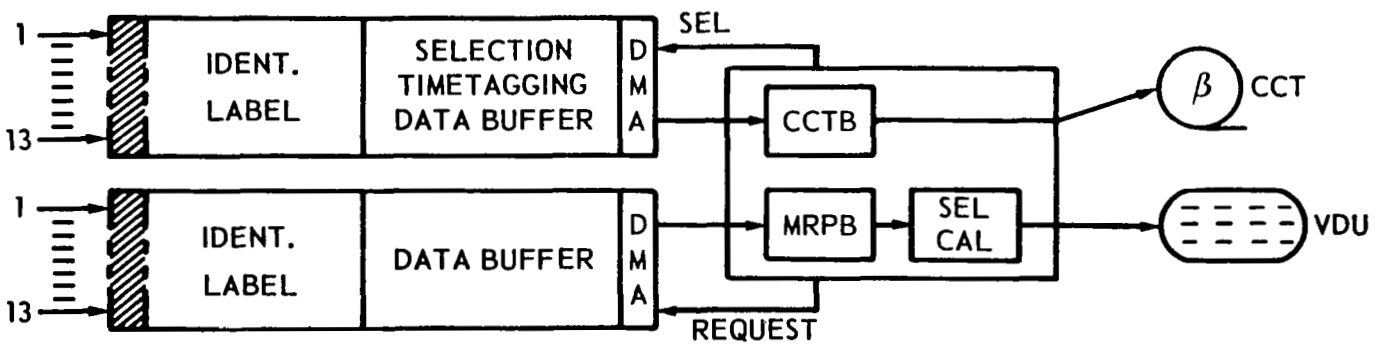


100 % CPU LOAD FOR INPUT , IMPOSSIBLE

Figure 15

OBC DESIGN

PHASE II MORE FUNCTIONS IN HARDWARE



FIXED FUNCTIONS IN HARDWARE

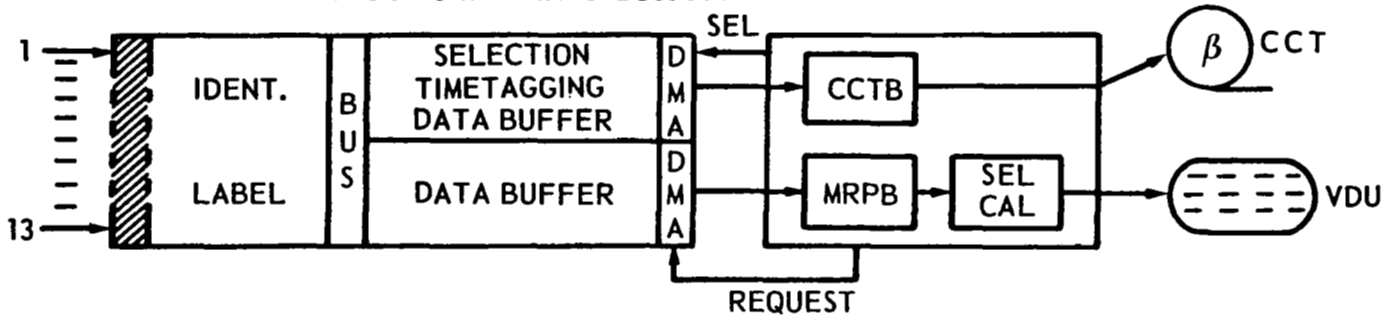
ALTERABLE FUNCTIONS IN SOFTWARE

HIGH HARDWARE COST (DUPLICATION OF I/O CIRCUITS)

Figure 16

OBC DESIGN

PHASE III **COMBINING I/O, IDENT AND LABELING INTRODUCING INTERNAL BUSSYSTEM**



PREPROCESSOR WITH SELECTION, TIMETAGGING AND DATABUFFERS.
NUMBER OF I/O CIRCUITS MINIMIZED

Figure 17

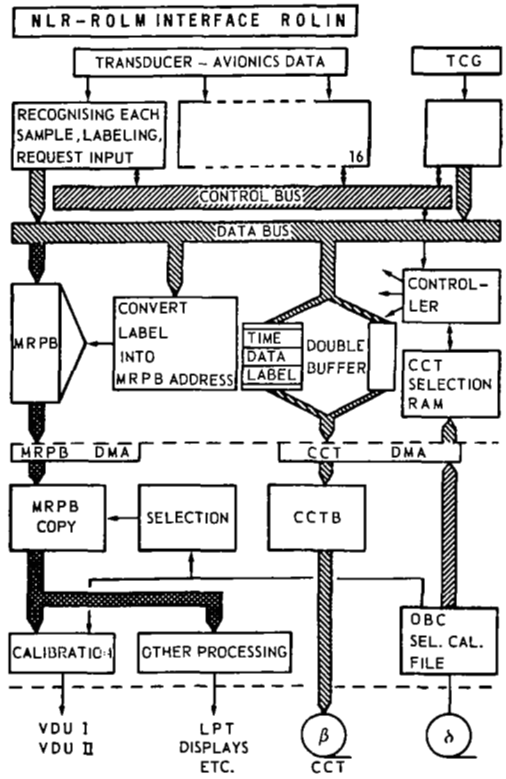


Figure 18

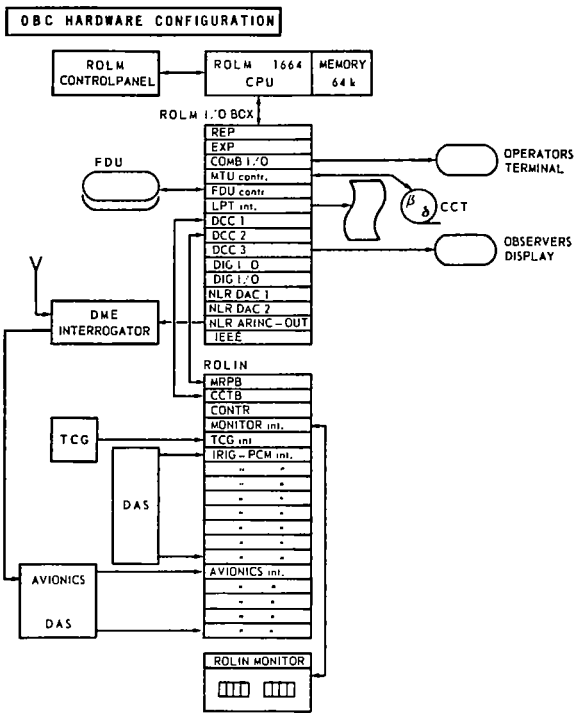


Figure 19

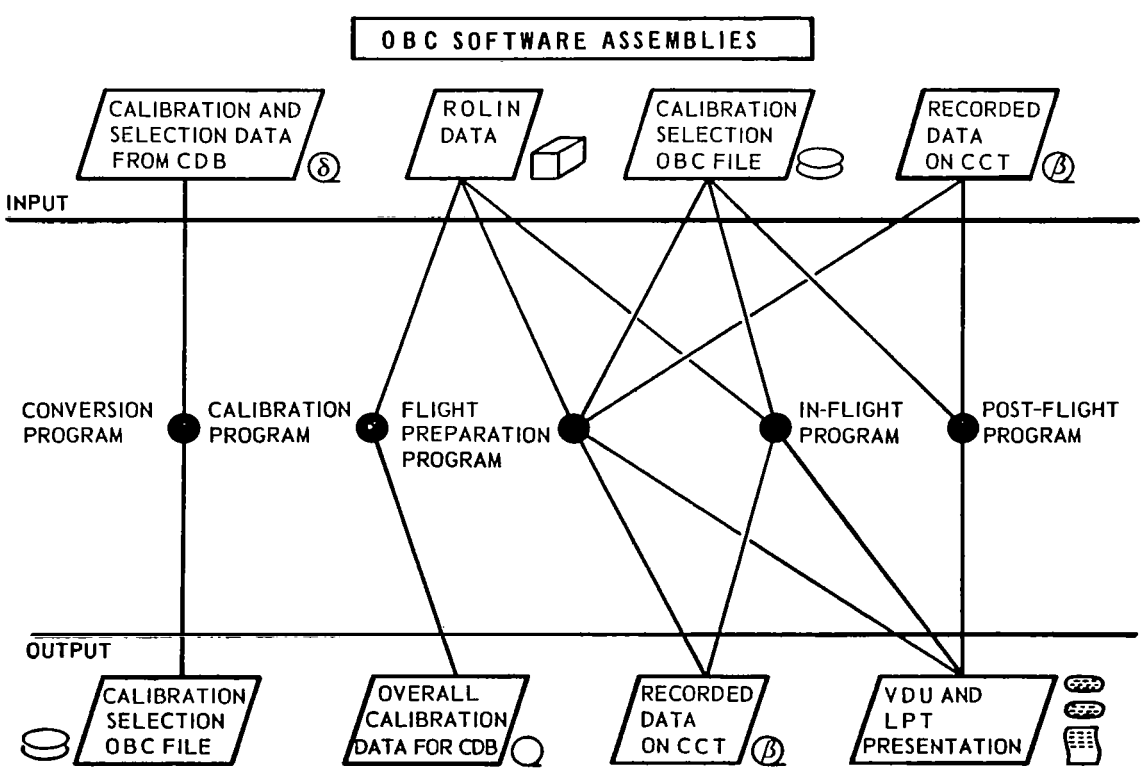


Figure 20

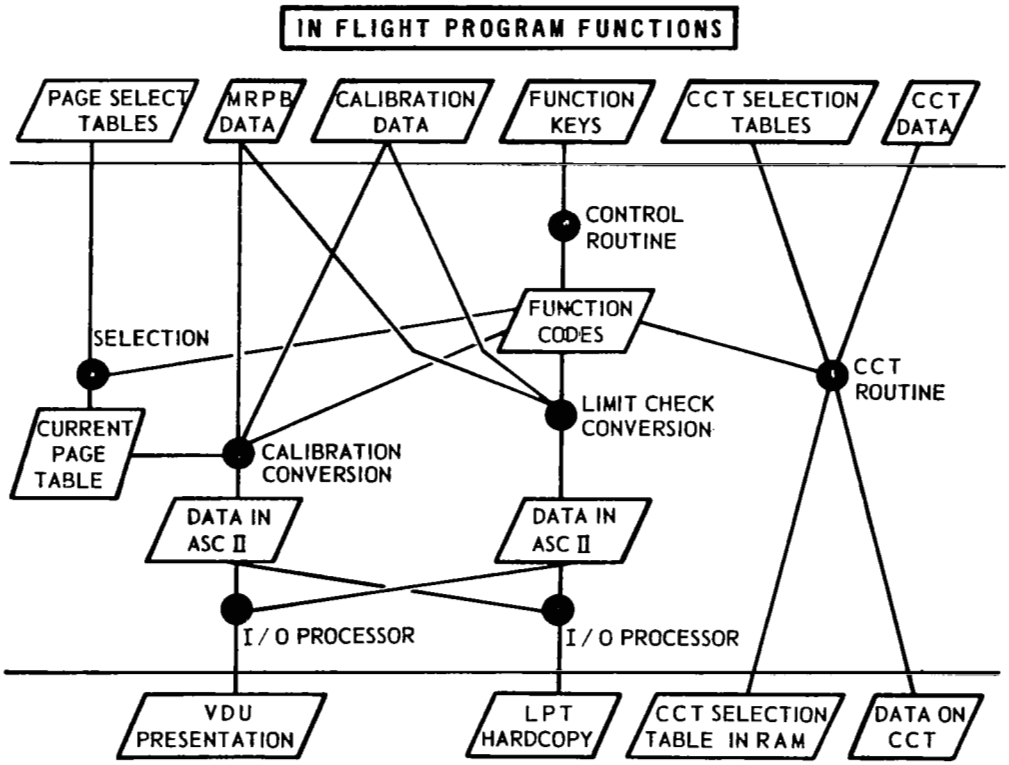

IN FLIGHT PROGRAM FUNCTIONS
PAGE SELECT TABLES
MRPB DATA
CALIBRATION DATA
FUNCTION KEYS
CCT SELECTION TABLES
CCT DATA
CONTROL ROUTINE
FUNCTION CODES
SELECTION
CCT ROUTINE
LIMIT CHECK CONVERSION
CURRENT PAGE TABLE
CALIBRATION CONVERSION
DATA IN ASC II
DATA IN ASC II
I / O PROCESSOR
I / O PROCESSOR
VDU PRESENTATION
LPT HARDCOPY
CCT SELECTION TABLE IN RAM
DATA ON CCT

Figure 21

FAULT ISOLATION TECHNIQUES

Al Dumas
Westinghouse
Baltimore, Maryland

ABSTRACT

There are four items which a user must take into consideration in the area of user maintenance of computer equipment:

(1) Establishing a philosophy
(2) Capitalizing maintenance levels
(3) Integrating with company policy
(4) Implementing a program

Because down time is expensive, users should focus on reducing it with a quick react approach to a system fault, or what appears to the programmer to be a system fault. This paper briefly describes three major areas that should be considered in the development of an overall maintenance scheme.

AREAS OF CONCERN

There are three areas of concern related to fault isolation techniques:

(1) The programmer (or user)
(2) Your company and its policies
(3) ROLM, the manufacturer of the equipment

Regarding the first area mentioned, service personnel often can diagnose actual machine problems as well as programmer problems. By using short sections of machine language coding the user can almost always at least differentiate between machine problems and programmer problems. It would be helpful to require the programmer to indicate in a log book the machine condition at the time of failure, in order to try to determine a sequence of events that may help the next time a failure occurs. One example of this is the position of the floppy disk selector, where there is more than one floppy drive in the system. The last programmer or operator may have left Unit A in "Select position 1" and Unit B in "Select position 0". When the next user tries to use Unit A as "DFO", nothing happens; the programmer or operator gets no input from that device since the select switch is improperly set up.

The second area of interest is your company's policies regarding repair of equipment. Some possible policies include:

(a) Calling the local service representative, if the machine is under a service contract

(b) Calling the lead programmer to determine whether it appeared to be a machine failure or a programming problem (i.e. a new programmer might not know system capability)

(c) Having knowledgeable people on board to service equipment

(d) Stocking a limited number of spares as suggested by ROLM. The user must determine to what level spares should be stocked (i.e. boards, chassis, power supply, etc.)

(e) Establishing a procedure according to the company's needs:

(1) This type of equipment must always be available, therefore redundancy must be either built in or available (i.e. a second system)

(2) All items that can be replaced in the field shall be stocked as spares, as suggested by ROLM, and the company will have qualified technicians

(3) The company wants service upon request; it does not want to maintain spares or qualified technicians

(4) Down time is not a major concern; the company will return system or suspected unit for repair

Some of the company policies mentioned affect all users and therefore are offered as guidelines. Users who are relatively new at computer systems maintenance may not be aware of all the pitfalls associated with computers.

Because down time can be more costly to a company than service costs or spares, the following guidelines are offered as a way of determining the user's relative costs.

(1) Limited operation. Consider whether this operation can continue without certain components, such as a line printer or a disk.

(b) Limited capabilities. For instance, the system will operate if loaded from floppy diskettes, but the magnetic tape is not available because of an I/O board failure, and the terminal causes interrupts during extremely long listings and must output everything to the line printer.

(c) Minimum down time allowed. The system or any part of the system can never be down for more than one hour because other people are affected.

(d) Down time cost factor. For every hour the system is down it puts six people out of work at $5.00 per hour = $30.00 per hour.

(e) Machine use. This machine must be available 8 hours a day, 5 days a week. This means the system is available for maintenance 16 hours a day × 5 days a week = 80 hours + 48 hours on weekends = 128 hours of allowable down time.

The final area of concern in establishing an overall maintenance scheme is determining what information the equipment manufacturer can provide which would be helpful in isolating machine-related problems. Some manufacturers supply users with a flow diagram of fault isolation procedures to the board level (fig. 1). A user's guide for fault isolation, similar to the operator's handbook used by programmers, would also be helpful. Such a manual might include:

(a) A section on manual loading of programs (i.e. to test the real-time clock when diagnostics cannot be loaded via any other media)

(b) A section on diagnostic program instructions (i.e. place tape on PTR, set data switch to all ones, press START. Tape loads, set data switch to 52, depress START)

(c) A section on type of HALT as indicated by display panel

(d) Recommended repair procedure

Users can help one another by exchanging information on fault isolation techniques which have been developed. This exchange can be implemented as a portion of annual users group meetings, such as this one, with the user and manufacturer diagnostic techniques documented in a conference proceedings. An alternate technique might be for the manufacturer to collect diagnostic techniques contributed by many users and publish the resulting collection in a group newsletter. The specific procedure used to exchange information is not critical; the important thing is to actually share these valuable experiences.

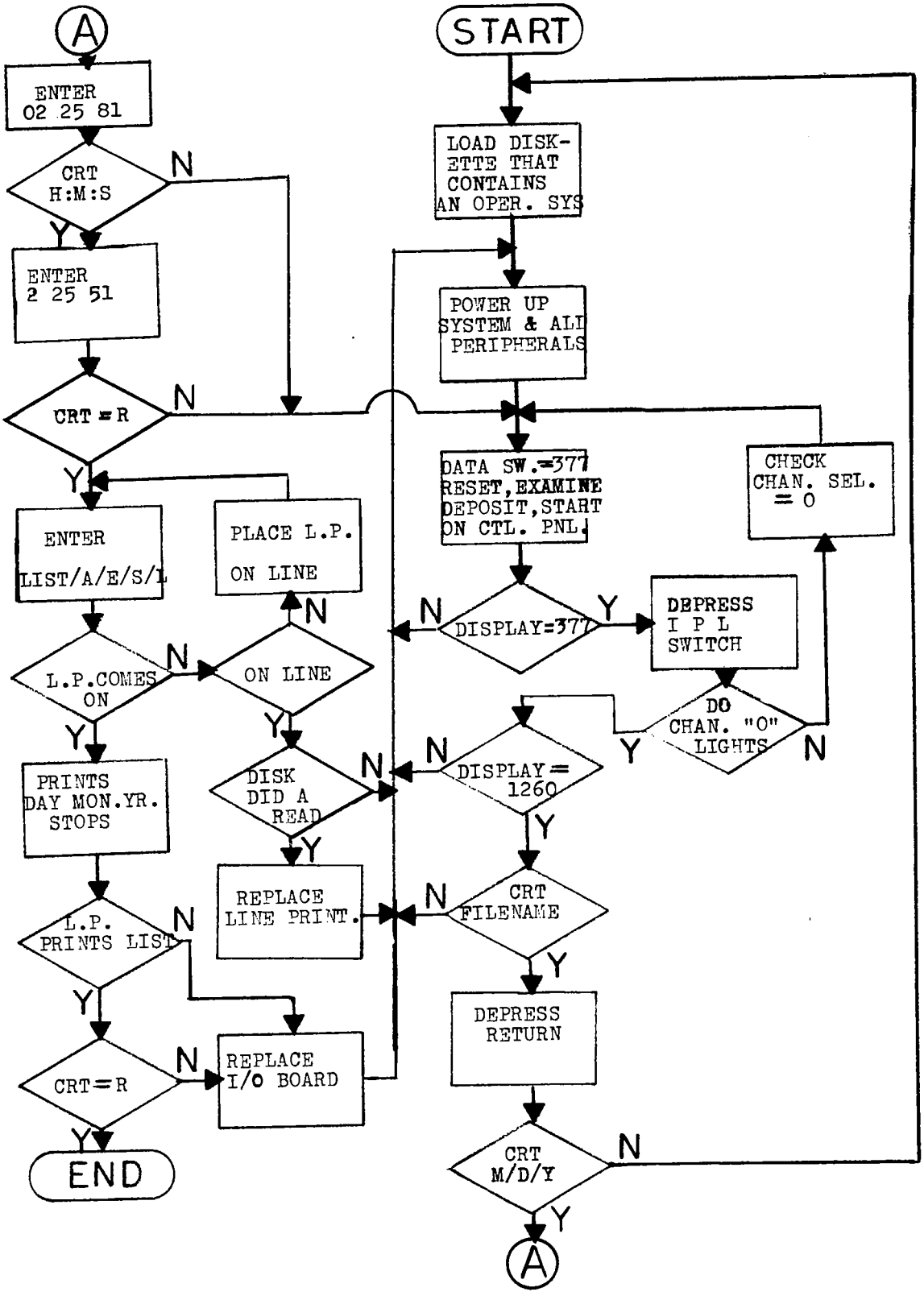

A
ENTER
02 25 81
CRT
H:M:S
N
Y
ENTER
2 25 51
CRT = R
N
Y
ENTER
LIST/A/E/S/L
L.P.COMES
ON
N
Y
PRINTS
DAY MON.YR.
STOPS
L.P.
PRINTS LIST
N
Y
CRT=R
N
Y
END
PLACE L.P.
ON LINE
N
ON LINE
Y
DISK
DID A
READ
N
Y
REPLACE
LINE PRINT.
REPLACE
I/O BOARD
START
LOAD DISK-
ETTE THAT
CONTAINS
AN OPER. SYS
POWER UP
SYSTEM & ALL
PERIPHERALS
DATA SW.=377
RESET,EXAMINE
DEPOSIT,START
ON CTL. PNL.
N
DISPLAY=377
Y
DEPRESS
I P L
SWITCH
DO
CHAN. "0"
LIGHTS
Y
N
CHECK
CHAN. SEL.
= 0
N
DISPLAY=
1260
Y
CRT
FILENAME
N
Y
DEPRESS
RETURN
CRT
M/D/Y
N
Y
A

Figure 1

EXTENDED MEMORY MANAGEMENT UNDER RTOS

USING FORTRAN

Mark Plummer
GTE Sylvania
Mountain View, California

ABSTRACT

Direct user access of extended memory in the ROLM 1666 is not supported under RTOS. Extended memory pages must be mapped to a window in the user program before they can be accessed. The amount of extended memory usable at any moment is thus limited by the size of the window. A Memory Management Module (MMM) has been developed which, despite this limitation, manages dynamic memory allocation with a minimum window size. The MMM is designed to support large data buffers that are accessed infrequently by multitask user programs. It is useful in systems where data buffers spend large amounts of time on I/O or communication queues. Use of the MMM makes extended memory manipulations transparent to FORTRAN user programs.

1.0 INTRODUCTION

This paper presents a technique for extended memory management in ROLM 1666 computers using FORTRAN. A general software system is described for which the technique can be ideally applied. How the memory manager interfaces with the system is described in detail. The protocols by which the manager is invoked are presented, as well as the methods used by the manager. Several problems associated with the technique are discussed.

Terms used throughout this paper are defined as follows:

A system is composed of many segments (tasks), both independent and related, called modules. The memory manager is itself a module. Those modules which invoke the manager will be called "user modules" (Fig. 1). Logical address space is the amount of computer memory addressable by a user module; in the case of the ROLM 1666 this is 64K words. Extended memory is that memory which exceeds the logical address space (Fig. 2). A "window" is an area of addressable space which can be mapped to an area in extended memory. RMX/RTOS performs mapping on 1K-word pages, hence windows must be in sizes which are multiples of a page. Any page in addressable space can be defined as a window and mapped to any page in extended memory. The page to which a window is mapped is said to be "user accessable"

(Fig. 2). The memory manager described in this paper uses the term "buffer" as any contiguous section of memory which it is managing. An "active" buffer is one which is user accessable. A "deactive" buffer is one which is not presently user accessable (Fig. 2).

2.0 SYSTEM SCENARIO

It is not uncommon for a FORTRAN software system to process data occupying large amounts of memory during each execution cycle. If the program itself is large it may be necessary to store most of the data in extended memory. Under RMX/RTOS all extended memory references must be made to a window in logical address space, which is mapped to extended memory. The amount of extended memory accessable at one time is therefore limited by the size of the window. This also holds under RMX/RDOS for all operations except disk I/O.

Frequently in such a system only a small amount of the data must be accessed at any given moment. Often there are several independent system modules which must access the data. The modules may be driven by external events, such as inter-computer I/O, disk I/O, and other peripheral device I/O. They are thus accessing and processing the data asynchronously. Most modules only need to acquire or set a specific element of a data buffer, using it for a very short period. It is then passed to another module or I/O device, often on a communications queue (Fig. 3). When so transmitted it is not necessary for the data buffer to be accessable. A window for extended memory therefore need only be as large as the maximum amount of data being accessed at one time. Due to the staggered timing of the modules and the use of queues, this is generally only several pages of memory. A data buffer is mapped into this window only when it is immediately needed. At all other times it is mapped into extended memory, freeing the window for other users. For example, in a system which processes digital audio data and can receive a burst of 20 1K word buffers of input data in approximately 100 msec, a window size of only 5 pages is required.

3.0 THE MEMORY MANAGEMENT MODULE (MMM)

A Memory Management Module has been implemented which allows multitask FORTRAN users of data buffers to access them only when needed, while keeping the actual mapping procedures transparent. The MMM can support data buffers whose sizes are any multiple (fractional or integral) of a page. The present implementation supports two sizes: ¼ page and 1 page. These sizes were chosen for uniformity with system data and disk sector size.

The RMX/RTOS window area used by the MMM is declared as a COMMON array titled "BUFFER", which is accessable by all system modules. The array BUFFER is dimensioned in multiples of a page, and must begin on a page boundary for compatibility with RTOS. The MMM supplies users with indices into this array.[1] The index locates the beginning of the supplied buffer in the window (Fig. 4).

The MMM indentifies data buffers by a unique key called the buffer "name" (Fig. 5). The buffer name is supplied to users by the MMM upon request for a buffer. Since the index of each buffer will change between activations, the name must be maintained by users and supplied to the MMM for all other buffer operations. No other means of identifying a buffer exists.

3.1 Initialization

Preceding any user requests to the MMM, initialization must be performed by the following call:

```
CALL MEMIT
```

This call initializes a table internal to the MMM which is the heart of the module (Fig. 6). The table contains an entry for each 1K-word page in both the window and extended memory. Associated with each table entry is the logical page number in the window to which the page is mapped if the page is active, or a flag to indicate that the page is deactive. The buffer name is an index into this table, plus an index into the page if a ¼-page buffer (Fig. 5). An assigned/unassigned status flag is also maintained in the table for each buffer in the page. During initialization the maximum number of pages in the window are flagged as being active and all others are deactive.

3.2 Get a Buffer

A user module which wishes to acquire a data buffer must execute the following call:

```
CALL MEMGET(buffer size, buffer index, buffer name,
            error return)
```

The buffer size requested must be in the range 1 to 1024 words, although the returned buffer is either 256 or 1024 words. The buffer name must be maintained for future MMM operations. The buffer may be transmitted to another module, via a direct call or communications queue, by passing the name. If the buffer is to be placed on a queue the user module should deactivate it. The buffer supplied to the user by the MMM is located in

```
BUFFER (buffer index) to BUFFER(buffer index +
                                returned buffer size -1)
```

(Fig. 4).

Upon receiving this call the MMM searches its internal table for an unassigned buffer which is mapped into the window. If none exists the error return is taken, informing the caller that no buffer is available. Otherwise the appropriate buffer name and buffer index are calculated and returned. The MMM buffer allocation scheme must attempt to reduce fragmentation within memory. This is presently done by allocating 1-page buffers from one end of the table and ¼-page buffers from the other.

3.3 Release a Buffer

A user module which has completed all processing of a data buffer can release it for re-use with the following call:

```
CALL MEMREL(buffer name)
```

This call causes the MMM to flag its table entry for the specified buffer as unassigned, regardless of the active/deactive status of the buffer. An additional check is made to flag the entire page as unassigned if the buffer released was the last in a page of ¼-page buffers.

3.4 Deactivate a Buffer

A user module which wishes to temporarily release access to a buffer does so with the following call:

CALL MEMDACT(buffer name)

The buffer index held by the user after this call is no longer valid for that buffer and must not be used.

This call causes the MMM to search its table for an unassigned page in extended memory (a free, deactive page). If one is found, it is swapped with the supplied buffer. The free page thus becomes active and available for future get-buffer operations. The supplied page becomes deactive. Specifically, the MMM interchanges the table entry of the free page with the one specified by the input buffer name. After thus flagging the map in its table, an RTOS call is executed to perform the actual operation. This puts the new, unassigned page into the window. If the buffer deactivated is of size ¼ page, the page containing it is mapped only if there are no active ¼-page buffers remaining in that page.

3.5 Activate a Buffer

A user module which needs to access a buffer that has been deactivated must first execute the following call:

CALL MEMACT(buffer index, buffer name, error return)

The buffer name is supplied by the user module and the buffer index of the activated buffer is returned by the MMM. The buffer is used in the same manner as after a get operation.

Similiar to a get operation, the MMM searches its internal table for an unassigned and active page. If one is not found the error return is executed, informing the caller that there is no space for activation. Otherwise the appropriate table entries are interchanged and the RTOS map is performed. If the buffer being activated is a ¼-page buffer, the page must

be mapped regardless of the deactive or unassigned status of the remaining ¼-page buffers in the page.

The MMM operates in a multitask environment so it must be reentrant. This requires its internal table to be protected by semaphore locks. The RTOS intertask communication mechanism of XMT-REC can be used very effectively for this purpose. However, the present implementation of the MMM must operate in the interrupt handler environment as well as the task environment. Because of this, semaphore locks in the MMM are implemented by a test and set capability performed on lock locations in the internal table (Fig. 6). Only MEMGET and MEMREL calls can be issued from an interrupt handler, since the RTOS mapping calls in MEMDACT and MEMACT cannot.

4.0 LIMITATIONS

Due to the size of the system program the size of the window may be smaller than that required during peak buffer usage. This will affect get and activate operations. A feasible solution is a utility module which executes a delay and retry whenever a buffer is unavailable. The delay allows other modules to finish processing and release buffers, freeing the necessary window space.

Since user modules access their buffers through a COMMON array there is no way of insuring that they do not access other buffers. Care must be taken when programming modules which use the data buffers to prevent accessing out of bounds.

A major difficulty encountered in using the MMM in a large system is insuring that a buffer is released only once. The possibility exists that two separate modules may receive the name of a buffer containing a data block which they must both process. The system design must prevent both modules from releasing the buffer. If not, the second module to release the buffer would most likely be incorrectly releasing the buffer after allocation to an entirely different data block. A minimal but simple check for this error is to set the name of a released buffer to an invalid value. This insures that

if a buffer released by one routine is inadvertently transmitted to another, the invalid name will generate an error return. A more complete solution is to maintain a count of the number of times a buffer page has been allocated and include that count in the name (Fig. 7). This would make the buffer name unique for each data block to which it is assigned (up to the rollover of the counter), insuring only one release of a data block buffer per assignment.

5.0 SUMMARY

The MMM was designed for a specific system but fits the requirements of many real time, multi-task data processing environments. It is appropriate wherever large amounts of core-resident data must be handled and routed, but infrequently accessed. It is easy to interface with from FORTRAN. Due to the minimum window size required, it is especially useful in systems which have a limited area in addressable space for data buffers.

REFERENCE

1. Dowell, R.: Efficient Memory Usage on Mini-Computers Using FORTRAN. COMPCON 1980, Proc. of 20th IEEE Computer Society International Conf., 1980, pp. 70-72.

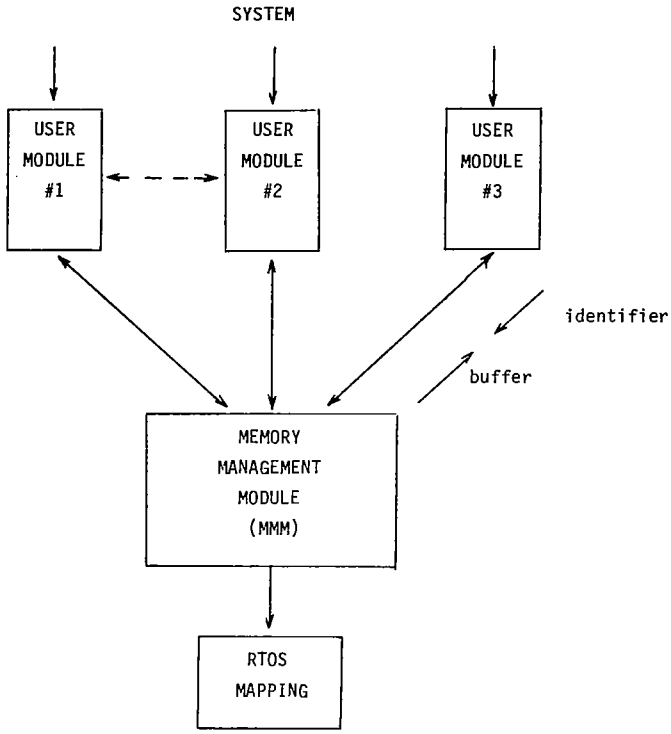


Figure 1

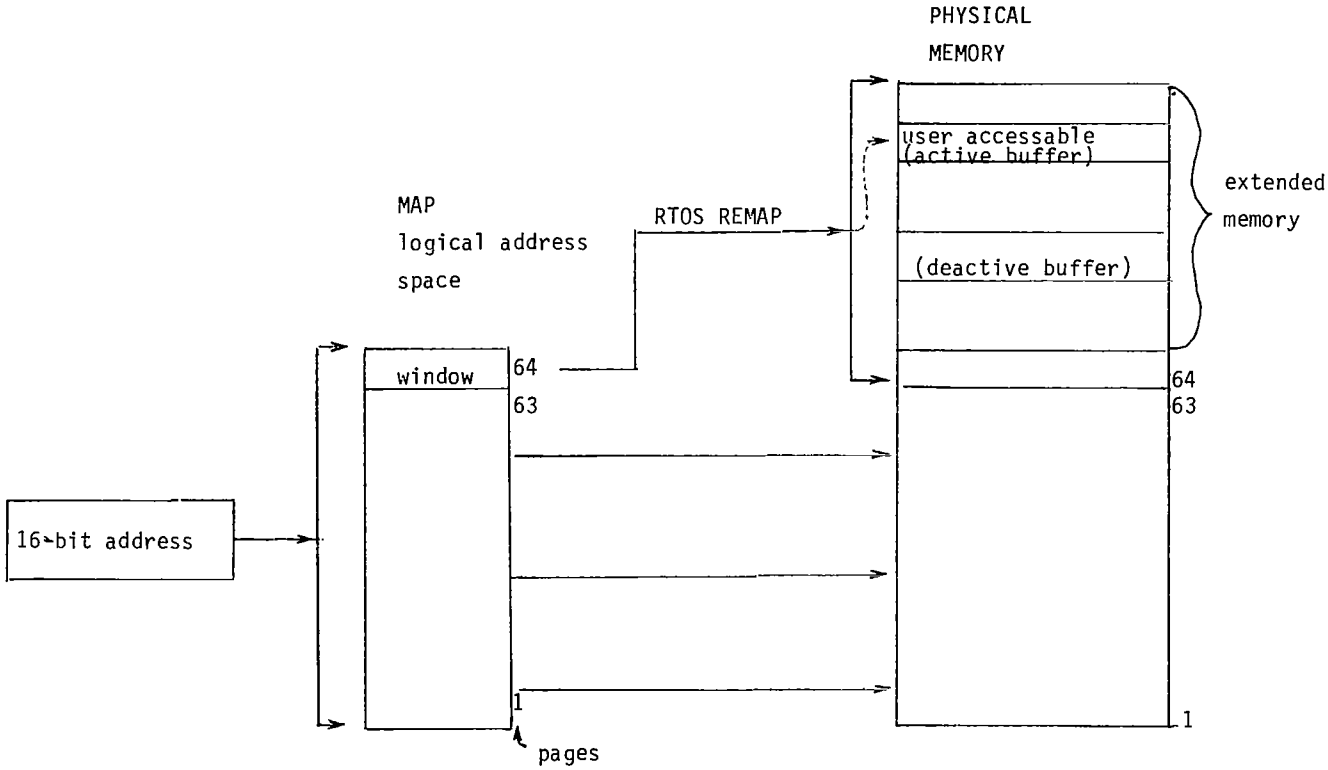


Figure 2

STORAGE

RETRIEVAL

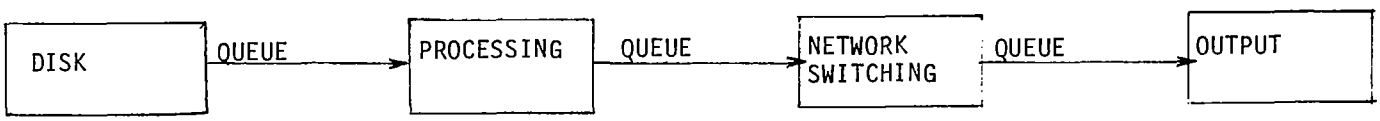


Figure 3

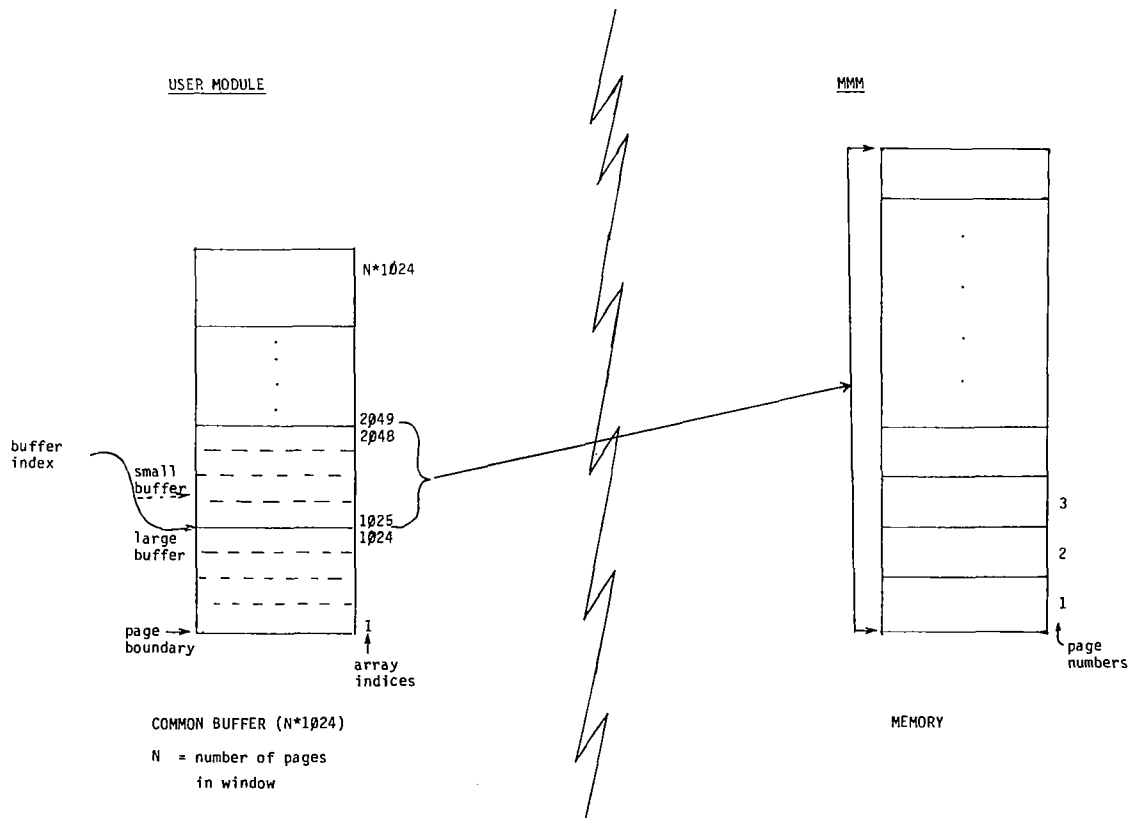


Figure 4

MMM BUFFER NAME

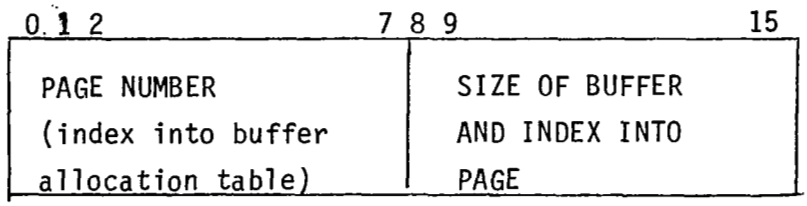
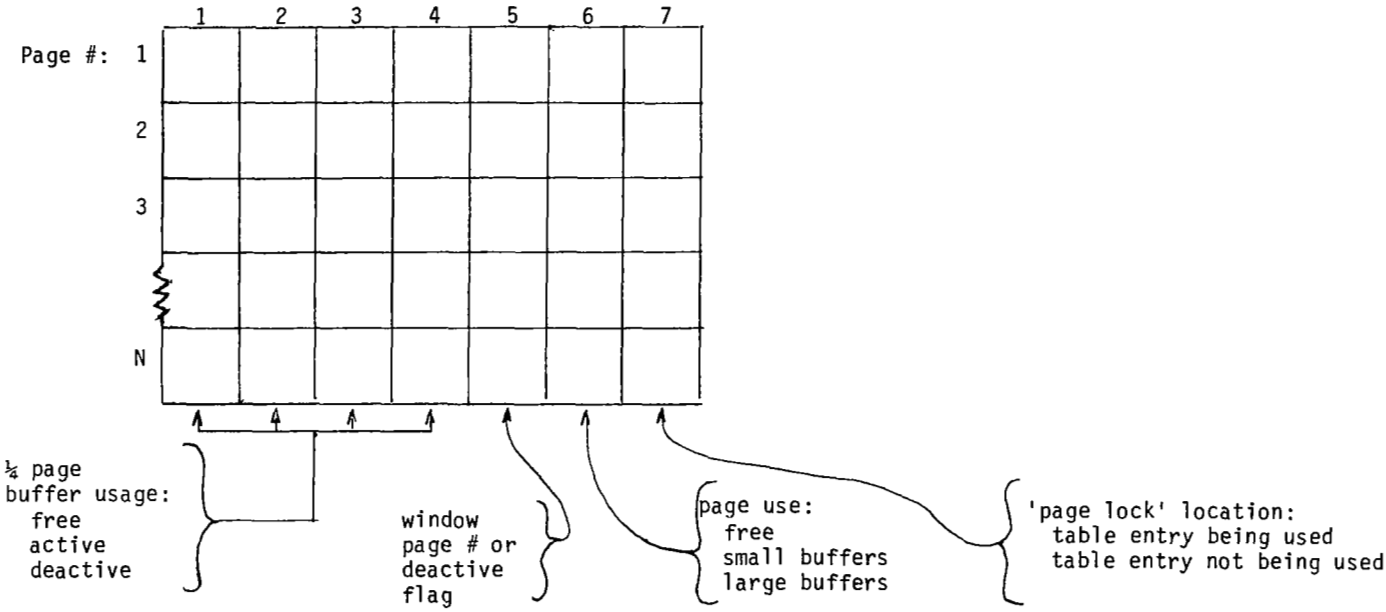


Figure 5

MMM BUFFER ALLOCATION TABLE



N = number of pages in window
+ number of pages in extended memory used for buffers

Figure 6

IMPROVED MMM BUFFER NAME

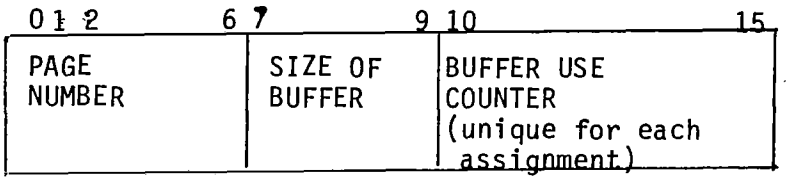


Figure 7

DESCRIPTION OF A DUAL FAIL-OPERATIONAL REDUNDANT STRAPDOWN INERTIAL MEASUREMENT UNIT FOR INTEGRATED AVIONICS SYSTEMS RESEARCH

W. H. Bryant and F. R. Morrell
NASA Langley Research Center
Flight Electronics Division
Hampton, Virginia 23665

Abstract

The general trend in modern aircraft is toward integrated, all-digital avionics systems. Langley Research Center of NASA is developing a research oriented dual fail-operational redundant strapdown inertial measurement unit which will be used to study such systems as a primary source of navigation, guidance, flight control,and display data for integrated avionics systems. This developmental system will be used to examine failure detection and isolation algorithms, and determine optimum failure thresholds at the sensor level, given mission constraints. The major emphasis is to ensure highly reliable data for flight control while minimizing false or missed alarms.

The redundant strapdown inertial measurement unit, which includes a skewed array of four two-degree-of-freedom gyros and accelerometers, is coupled through instrument electronics to two flight computers; this will provide a means to test and evaluate dual fail-operational concepts of inertial measurement units. The inertial sensors and compensation electronics form two separable non-redundant blocks to enable investigation of variable sensor location on the failure detection and isolation problem.

It is intended to demonstrate by means of laboratory and flight tests that a low-cost dual fail-operational strapdown system of sensors is capable of providing an improved integrated avionics function. This requires substantial visibility via the flight computers to the redundant strapdown inertial measurement unit system operation.

1. Introduction

Because of recent advances in computer technology, modern aircraft systems are nearing an all-digital status. Many analog systems have been replaced by digital systems which bear their own digital computer and interface requirements. The progress in digital computer technology has also spurred advancement in strapdown inertial sensors because significant cost savings result from integrated avionics systems for flight controls, navigation, and display.[1]

As aircraft incorporate advanced energy efficient design, they will become more reliant upon these integrated avionics systems to the point that selected avionics components/systems may become flight critical. The most obvious candidates are those systems which provide the aircraft stability, from sensors through actuators. These systems must have the inherent capability to provide augmented stability without adding to pilot workload by requiring that the pilot monitor sensor operations and switch out failed units.

Langley Research Center of NASA is developing an experimental redundant strapdown inertial measurement unit (RSDIMU) as a link to satisfy safety and reliability considerations in the integrated avionics concept.[2,3] The unit includes four two-degree-of-freedom (TDOF) tuned rotor gyros, and four TDOF accelerometers in a skewed and separable semi-octahedral array. These sensors are coupled to four microprocessors which compensate sensor errors. These microprocessors are interfaced with two flight computers which process failure detection, isolation, redundancy management,and general flight control/navigation algorithms. Since the RSDIMU is a developmental unit, it is imperative that the flight computers provide special visibility and facility in algorithm modification.

2. Description of the RSDIMU

The redundant strapdown inertial measurement unit has a complement of instruments mounted in a semi-octahedral configuration (Fig. 1) such that fail-op/fail-op capability is provided. When coupled with flight computers, a self-contained, inertial navigation system results.[2] The spin-axes of the TDOF gyros and the pendulous axes of the TDOF accelerometers (S_i) are normal to the faces of the semi-octahedron as shown in Fig. 2. The measurement axes of the gyros and accelerometers (Xi,Yi) are nominally collinear and are oriented such that the bisector of the angle between the sensitive axes is perpendicular to the baseline of the semi-octahedron (Fig. 3). The IMU sensor head is separable consisting of two self-contained non-redundant systems A and B as shown in Fig. 4. This design provides the flexibility required when investigating inertial sensors separation effects on the performance of the dual fail-operational system in the flight environment.

The instrument clusters use self-calibration at each system startup to compute gyro and accelerometer long-term instabilities. Based on known Earth-rate and gravity for two positions of the individual sensor heads, gyro and accelerometer bias, gyro g-sensitive and g-insensitive terms can be determined. Each instrument cluster contains two gyros, two accelerometers, and two calibration heads. The mounting blocks each have a precision surface and pins for accurate realignment after separation. The instrument electronics assemblies contain a microprocessor for each gyro-accelerometer pair,

gyro and accelerometer torquing electronics, analog-to-digital converters, and interface capability to transfer sensor information to flight computers. Each instrument electronics assembly is housed in an ATR box. Housed in a separate ATR box are the power conditioning system, 28 volt batteries and a charger to maintain the system operation for periods up to 20 seconds during power transfers. The RSDIMU sends gyro and accelerometer data, resolved along the ideal instrument-fixed coordinate reference frame, to the external flight computers. After processing this data, the flight computers return to each IMU microprocessor spin-axis and Earth-rate information for sensor compensation purposes.

As shown in Fig. 5, the sensor data output of the RSDIMU will be processed in two independent solutions corresponding to the IMU structure. This will allow examination of dual fail-operational performance at the sensor level. The data will be processed through failure detection/isolation and redundancy management algorithms, least squares solutions of sensor combinations, attitude update, and navigation/alignment functions. Sensor processing will occur at a 64 Hertz update rate.

3. General Description of RSDIMU Components

Inertial Package

The angular rate sensors are Incoflex two-axis dynamically tuned gyros. The steady state rate capability is in excess of 100 degrees/second; this single range torquing will reduce scaling errors due to switching between ranges, reduce software requirements, and considerably simplify the gyro torquing electronics. The RSDIMU gyro performance parameters given in Table 1 are commensurate with 2 Km/hr. navigation systems.[3]

Table 1. INCOFLEX GYRO PERFORMANCE PARAMETERS

ACCURACY IN deg/hr 1σ	
BIAS REPEATABILITY	0.01
RANDOM DRIFT	0.01
ANISOELASTICITY IN deg/hr/g^2	0.02
TORQUING RATES IN deg/sec @ 74°C ambient	
STEADY STATE	110
TRANSIENT (1 second)	220
% DUTY CYCLE @ 400 deg/sec	1
TORQUING ACCURACY IN PPM	50
PHYSICAL CHARACTERISTICS	
SIZE - DIA. X LENGTH IN cm	5.1 x 4.6
WEIGHT IN Kg	.34
SPIN SPEED IN RPM	12,000
POWER REQUIREMENTS IN watts	
SPIN MOTOR	2.0
TORQUER @ 1 rad/sec	2.0
PICKOFF	0.2

The linear acceleration sensors for the RSDIMU are Incoflex two-axis pendulous accelerometers. The two-axis suspension system for the accelerometer is a slightly modified version of that used for the Incoflex gyro. The torquer is pendulous relative to both torsional axes of the suspension system. Deflection of the pendulous mass is sensed by pickoffs and is proportional to the applied acceleration. Since the gyro and accelerometer axes are aligned on each face of the semi-octahedron, it is possible to package the sensors as shown in Fig. 1. The accelerometer performance characteristics are given in Table 2 and provide a navigation accuracy consistent with the gyro.[3]

Table 2. INCOFLEX ACCELEROMETER PERFORMANCE PARAMETERS

Range	10 g
Bias stability:	
long term	100 μg 1σ
two-hour drift	5 μg 1σ
Scale factor:	
stability, long term	50 ppm
non-linearity	0.1 μg/g^2
Temperature sensitivity over 65°C range:	
bias (uncompensated)	18.0 μg/°C max.
bias (compensated)	1.8 μg/°C max.
scale factor (compensated)	3.6 ppm/°C max.
scale factor (uncompensated)	420 ppm/°C
Anisoelastic cross-coupling	0.25 μg/g^2

Sensor Compensation Microprocessors

As illustrated in Fig. 5, each gyro/accelerometer pair has a microprocessor (Intel 8086) to allow for system growth. In the system being fabricated for flight testing two Intel 8086 microprocessors are installed for each half of the IMU. One is operated as the CPU master and the other operates in the slave mode. Each processor has its own memory, and each is assembled on its own board. The Intel 8086 is a 16-bit processor capable of performing signed and unsigned arithmetic operations in binary and decimal formats, including multiply and divide. Through use of a memory segmentation technique, each can directly address up to one million bytes of memory. Both units used here are operated at 4 MHz.

The internal architecture of the 8086 consists of two asynchronous processors, one to control the basic processor operation and bus, and the other to prepare and operate on the data. No input/output lines are dedicated on the CPU, but the bus can provide all the necessary control lines to the various peripheral equipment. The 8086 is well suited for multiprocessor configu-

rations because it uses its Lock signal to support a read/modify/write sequency and a Test signal for external processor synchronization.

When both halves of the RSDIMU are operated, a clock signal is transferred from one to the other so that they operate synchronously. This ensures the time-homogeneous transfer of data from the microprocessors to the interface with the flight computers.

Flight Computers

The flight processors used in this system are ROLM 1666 MIL-SPEC computers[4]. These machines are state-of-the-art general purpose 16-bit minicomputers packaged for applications requiring high reliability in hostile environments. The ROLM 1666 is a mapped, tri-processor machine featuring a microprogrammed multi-accumulator General Purpose Processor (GPP), a high-speed variable precision hardware Floating Point Processor (FPP), and a Direct Memory Access (DMA) processor. The GPP typically manipulates 16- or 32-bit operands, performs the setup for floating point instructions, and carries out programmed input/output instructions. The FPP can accommodate single precision (32-bit, 6 to 7 significant decimal digits), extended precision (48-bit, 12 to 13 decimal digits), and double precision (64-bits, 17 to 19 significant decimal digits) operands. The DMA processor enables high speed transfers (approaching one megaword/second) between main memory and an external device to occur simultaneously with program execution without program intervention.

In addition to its high reliability and excellent hardware capabilities, the ROLM 1666 is supplied with a variety of software[5] that makes this computer system an excellent choice for research applications. The vendor supplied software includes operating systems, compilers, assemblers, editors, file management systems, etc. The combination of excellent support software, high processor speed, and the number of peripheral devices that can be readily integrated into the computing system permits the systems engineer to carry out Failure Detection and Isolation/Redundancy Management algorithm research more effectively by providing much needed visibility into the inertial system operation. In addition, algorithm selection and modifications required to improve FDI/RM techniques are easily incorporated because of the ROLM system capabilities.

ROLM 1666-to-IMU Interface

The Intel 8086 microprocessors provide scaled, fixed-point, compensated sensor data, status information, and other data, to the ROLM computers using high-speed bit-parallel, word serial interfaces. Each word transmitted between the ROLM computer and the 8086 microprocessor is 16 bits wide. The transmitted data is represented by either one or two words (16 or 32 bits).

To implement the high speed interface between the IMU and the ROLM processors, a ROLM Data Channel Controller (DCC) was selected. The DCC is fabricated on a single modular printed circuit card and is contained in a ROLM input/output chassis. The Intel 8086, contained in the IMU electronics box, communicates with the DCC over signal cables 10.7 m (35 feet) long. The DCC provides the required control logic for block data transfers between the Intel 8086's and the ROLM 1666 memories, and uses the DMA capability of the ROLM processor. The use of the DCC in this application is described more fully in a later section describing the software mechanization.

Figure 6 is a block diagram of the main processing system components currently envisioned to conduct RSDIMU flight experiments. The interface signals between an 8086 and the ROLM I/O chassis containing a DCC are shown in the upper left part of the figure. Data is transferred between the ROLM and Intel systems using the appropriate set of 16 data lines. The Mode status line informs the 8086 of the DCC's expected direction for data transfers (either in or out). The Data Channel Active line indicates when the DCC is in a condition to effect data transfers; the Data Transfer Request line (Device Done) is pulsed by the 8086 to actually initiate each word transfer. The Data Transfer Request Acknowledge line (Input Ack. or Output Ack.) informs the 8086 that the requested transfer has occurred.

Figure 6 also shows the peripheral and other equipment used with the ROLM computer for this experiment. Each computer has its own control panel which permits examination/modification of memory and accumulators at the most fundamental level (octal data and machine instructions), and controls the start/stop program sequence. The terminal is a standard Texas Instruments Silent 700 which is used to initialize the navigation algorithm and other program constants, as well as log the RSDIMU system performance in real time. The tape recorder, a Genisco ECR-10 1/2-inch cartridge system, is used to store various sensor head parameters determined at startup as well as the navigation systems' performance. The FDI/NAV Control Unit, designed and built at NASA-Langley, is used for navigation algorithm mode control, real-time floating point display of pre-selected quantities, and the injection of simulated sensor failures for evaluation of the redundancy management and failure detection/isolation algorithms. The FDI/NAV Control Unit is interfaced to ROLM computer "A" as shown in Fig. 6 and uses the ROLM 16-Bit Parallel I/O Buffer for data transfer. Data is passed between ROLM processors using a pair of 16-bit NTDS (Navy Tactical Data Systems, Standard DS-4772) interfaces, also manufactured by ROLM.

4. System Software Mechanization

The purpose of this section is to give an overview of the RSDIMU program flow. While the program has six modes of operation (Initialize, Coarse Align, Calibrate, Fine Align, Navigate, and Shutdown), the Navigate mode is typical of the task/sub-task and DCC Interrupt Handler (DCCIH) interactions and will be described here.

Figure 7 is a program flow diagram for the RSDIMU in the Navigate mode. The Main Task shown in the figure is created with a higher priority than the RSDIMU I/O Task, and begins its cycle waiting for a start message. This start message is issued by the DCCIH when an interrupt occurs after an IMU-to-Flight computer transfer has been completed. Since the IMU microprocessors are synchronized (Fig. 6) so that each will simultaneously transfer data to its respective flight computer at a 64 Hz. rate, the Main Task will execute at the same time in both flight computers; thus the IMU processors furnish the overall system synchronization.

The Main Task begins execution by converting 16- or 32-bit scaled integer values to their equivalent double-precision floating point engineering unit numbers. Two 16-bit status words accompanying each 8086-to-DCC transfer are decoded by this scaling routine to determine which set of scale factors should be used (and consequently which data have been transmitted). Transfers between the IMU and flight computers always consist of twenty-four 16-bit words so that the DCC setup is independent of program mode.

After the program in each machine converts the input values to their appropriate floating point equivalent, these converted numbers, the FDI/NAV command mode words, status words, and values to be recorded are passed between flight computers using the standard ROLM 16-bit NTDS interface. Since the FDI/NAV control unit is serviced by the "A" flight computer, the command mode word is passed only to the "B" computer. Similarly, data for real-time terminal display or tape recording is only passed from flight computer "B" to computer "A" (Fig. 6). At the completion of the transfers, both flight computers have the command mode words, status words, and sensor data for all sensor combinations; additionally, the "A" computer has all data to be recorded. The last operation carried out in this block is to implement simulated sensor failures according to the command mode word read from the FDI/NAV control unit.

After the coding represented by the first block has been completed, the Main Task flow depends entirely on the overall system mode selected (in this example, Navigate). The first step in the Navigate mode is the execution of Failure Detection and Isolation (FDI) and Redundancy Management (RM) algorithms. The development and evaluation of these algorithms is the main thrust of the current research effort and is indicated in Fig. 7 by the heavy block. One FDI/RM technique, which incorporates the edge vector method[2], will be described in more detail in a later section. After the FDI/RM algorithms are complete, the sensor attitude matrices are updated and the sensor spin- and pendulous-axes data is calculated for transfer to the IMU microprocessors. Finally, a standard navigation algorithm is executed and the Main Task is suspended until a message is posted by the DCCIH to repeat the sequence.

While the Main Task is suspended, the RSDIMU I/O Task executes. The FDI/NAV control unit is serviced once per second. Sixty-five sixteen-bit words are transferred to a buffer memory in the FDI/NAV control unit from flight computer "A". This buffer memory data is then continuously displayed on the FDI/NAV control unit as eight groups of four, eight-digit numbers (using seven segment displays) and one group of sixteen status indicators (discrete light emitting diodes). The status information is always presented, and a thumbwheel switch is used to select which group of four decimal numbers is displayed. Once the flight computer loads the FDI/NAV control unit buffer memory, the data group selection and display functions are carried out by the FDI/NAV control unit without flight computer involvement.

The hardcopy terminal routine is serviced next to provide data which permits the research engineer to evaluate the overall system performance in real time. Typical navigation data such as latitude, longitude, altitude, along with time of day are printed at various intervals ranging from 10 seconds to 60 seconds. Finally, combinations of system parameters are output to magnetic tape for post-flight analysis. The number of parameters recorded and the frequency of recording vary with the type of experiment being conducted. After the tape output routine is complete, the RSDIMU I/O Task begins its cycle again.

As noted above, one part of the Main Task requires the transfer of sensor compensation data from the flight computers to the IMU's. When this transfer is complete, an interrupt is generated by the Data Channel Controller and the flight computer transfers from the execution of the current task to the DCC Interrupt Handler. By recalling the last mode command word sent to the DCC, the DCCIH can determine whether the current DCC interrupt was generated after an input (to the flight computer) or an output (from the flight computer). In the present case, it is an output transfer and the DCCIH simply sets the DCC up for an input transfer and returns to the execution of the interrupted task.

At some later time (dictated by the IMU primary interrupt clock), data is transferred from the IMU to the flight computer and causes the DCC to generate another interrupt. The DCCIH then (1) determines that this interrupt occurred after an input transfer, (2) posts a message to start execution of the Main Task, (3) requests the operating system (O.S.) to determine which ready task has highest priority (reschedule), and (4) returns control to the O.S. for subsequent transfer to the Main Task. At the completion of this event, the program flow repeats the sequence previously described.

5. Failure Detection and Isolation/Redundancy Management Algorithms

The primary objectives of the experimental RSDIMU is to develop and evaluate FDI algorithms for dual-fail operational performance. For this system, it is required to survive two gyro and/or accelerometer failures. The addition of the ROLM flight computers to the RSDIMU provides the flexibility to simulate sensor failures through

the use of the FDI/NAV control unit.

The FDI system will be mechanized to detect and isolate three levels of failures[6]:

(1) Hard-failures: Large magnitude failures which affect flight control performance.

(2) Mid-failures: Medium level failures which affect pilot-display performance.

(3) Soft-failures: Low level failures which affect navigation performance.

Figure 8 is a block diagram of the FDI system processing. Hard-failures must not be allowed to propagate to the flight control system; therefore a hard-failure must be detected and isolated on the cycle in which it occurs. Mid-failure and soft-failure detection is processed more slowly after appropriate filtering to smooth the effects of sensor quantization and noise.

There are several ways to derive parity vectors for the semi-octahedron configuration[2,6]. The edge vector method is the simplest to implement. The measurements for two gyros or two accelerometers are compared along a vector, $\overline{E}_{ij}$, that is perpendicular to their spin axis vectors $\overline{S}_i$ (i=1,2,3,4). From the geometrical configuration indicated in Figs. 2 and 3, this vector lies along an edge of the semi-octahedron defined as

$$\overline{E}_{ij} = \frac{\overline{S}_i \times \overline{S}_j}{\left|\overline{S}_i \times \overline{S}_j\right|} \qquad j>i \qquad (1)$$

The parity equations for the gyros are defined as

$$P_{ij} = (\overline{W}_i - \overline{W}_j) \cdot \overline{E}_{ij} \qquad j>i \qquad (2)$$

where $\overline{W}_i$ is the output of the i^{th} gyro. For the fail-op/fail-op configuration there are six parity equations[7]. The parity residuals are tested against a threshold to detect failures. A logic variable, F_{ij}, is set true if the corresponding parity equation exceeds the failure threshold. The logic to isolate a failure in gyro 1 is,

$$F_{g1} = (F_{12} \cdot F_{13})\ (\overline{F}_{g2} \cdot \overline{F}_{g3}) + (F_{12} \cdot F_{14})\ (\overline{F}_{g2} \cdot \overline{F}_{g4}) + (F_{13} \cdot F_{14})\ (\overline{F}_{g3} \cdot \overline{F}_{g4}) \qquad (3)$$

There is a failure in gyro 1, indicated by F_{g1} set true, if any of the three pairs of parity residuals has exceeded the failure threshold and the corresponding gyros have not failed. When this method is extended to all four gyros, two failures can be isolated. One parity equation remains after two failures to detect but not isolate a third failure. On detection of a third failure, the system would cease operation. The accelerometer failure detection and isolation is determined in a similar manner.

The design of the RSDIMU when coupled with the flight computers, as indicated by Fig. 5, allows for two independent navigation solutions to be processed. It is possible to run more than two solutions simultaneously with proper least squares combinations of sensors. If a sensor in system A fails, system B parameters can be used to reset system A parameters to account for the cumulative effect of the failed sensor within the limitations of the system. Table 3 indicates the least squares combinations of sensors designed in the RSDIMU.

Table 3. LEAST SQUARES COMBINATIONS OF SENSORS

Failure Gyro/Accel.	Channel A	Channel B
0	1, 2	3, 4
1	2, 3	3, 4
2	1, 3	3, 4
3	1, 2	1, 4
4	1, 2	1, 3
1, 2	3, 4	3, 4
3, 4	1, 2	1, 2
1, 3	2, 4	2, 4
1, 4	2, 3	2, 3
2, 3	1, 4	1, 4
2, 4	1, 3	1, 3

To demonstrate the FDI and redundancy management capability of the RSDIMU software, the system was simulated over a trajectory of seven 90° turns at 76 m/s. The simulation included initial condition errors for alignment, velocity and position, and inertial sensor errors given in Tables 1 and 2 as well as random noise. During the flight, soft-failures (since these are the most difficult to detect) were inserted into the system according to the schedule of Table 4. The failures were simple bias shifts.

Table 4. SENSOR FAILURE INSERTION AND DETECTION

Failure	Sensor	Magnitude	Time Applied Seconds	Time To Detection Seconds
1	gyro 1	2.5°/hr	400	144
2	accel 1	5×10^{-3}g	1100	55
3	gyro 2	-3.5°/hr	1800	422
4	accel 2	-5×10^{-3}g	2500	239
5	gyro 3	3.5°/hr	3200	312

The effects of the sensor failures on velocity error for system A are indicated in Fig. 9. There are four resets of system A before system failure because of degradation in gyro 3. The failure detection levels for the gyro and accelerometer parity equations were set at 0.06° and 2.1 m/s respectively. Figure 10 shows the effect of gyro failures on the heavily filtered parity equations, and Table 4 gives the times of failure insertion and isolation.

6. Concluding Remarks

A redundant strapdown inertial measurement unit which will provide dual fail-operational performance has been designed and will soon become available for testing. The unit features state-of-the-art components with data processing frequencies consistent with integrated avionics concepts currently envisioned. Development of this unit will allow testing of failure detection, isolation and redundancy management algorithms in the flight environment.

References

1. Elson, Benjamin M.: 767 Digital Avionics Stress Flexibility. *Aviation Week & Space Technology*, vol. 109, no. 10, 1978, pp. 181-188.

2. Preliminary Design of a Redundant Strapdown Inertial Navigation Unit Using Two Degree of Freedom Tuned Gimbal Gyroscopes, NASA CR-145035, 1976.

3. Morrell, F. R. and Russell, J.: Design of a Developmental Dual Fail-Operational Redundant Strapped Down Inertial Measurement Unit. IEEE 1980 National Aerospace and Electronics Conference (NAECON 1980), volume 1, 1980, pp. 322-329.

4. Programmer's Reference Manual for the ROLM Model 1666 Processor. ROLM Corp., June 1977.

5. ROLM MIL-SPEC Computers Software Catalog. ROLM Corp., 1979.

6. Motyka, P.; Landey, M.; and McKern, R.: Failure Detection and Isolation Analysis of a Redundant Strapdown Inertial Measurement Unit, NASA CR-165658, February 1981.

7. Craig, R. J. and Russell, J.: Failure Modes and Redundancy Analysis for the Multifunction Inertial Reference Assembly (MIRA). AFFDL-TR-78-25, U. S. Air Force, 1978.

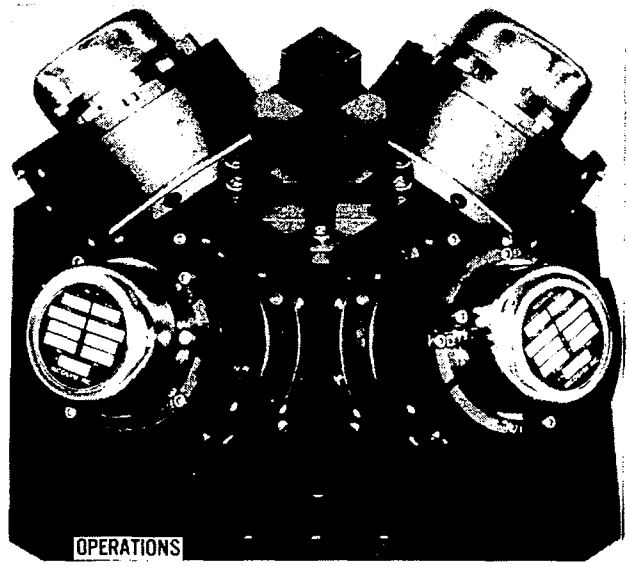


Figure 1. Semi-octahedron sensor mounting

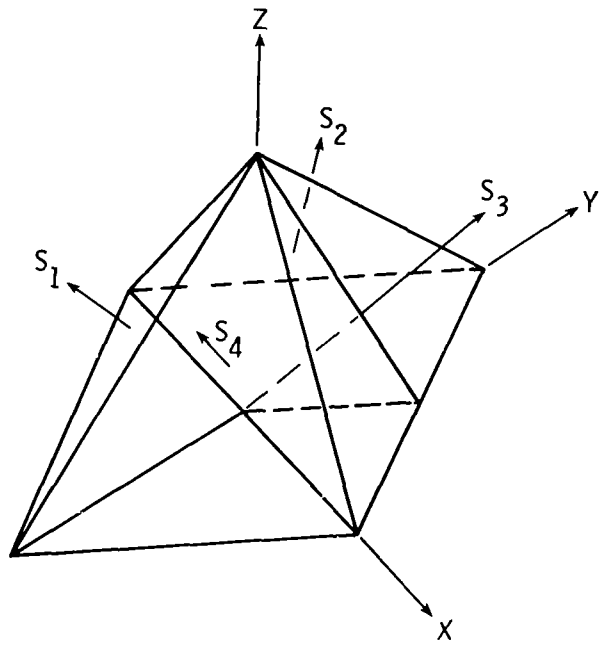


Figure 2. Sensor spin axis orientation

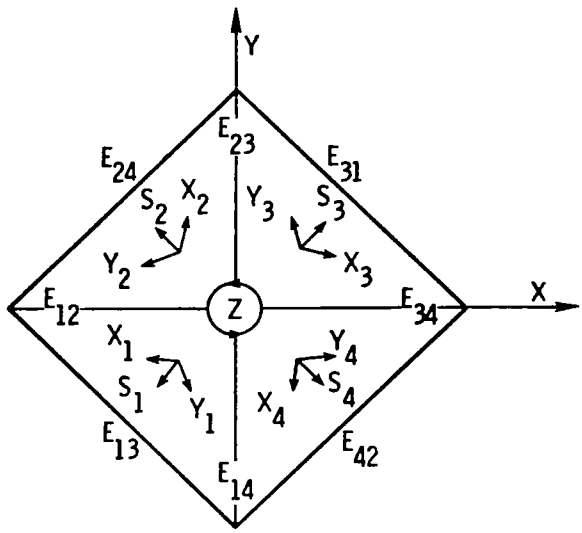


Figure 3. Sensor measurement axis and edge vector geometry

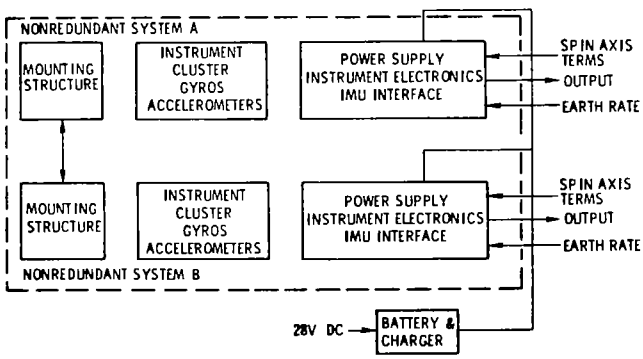


Figure 4. RSDIMU block diagram

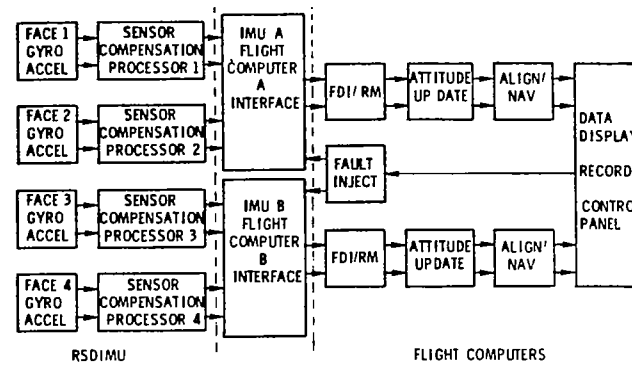


Figure 5. RSDIMU software data flow

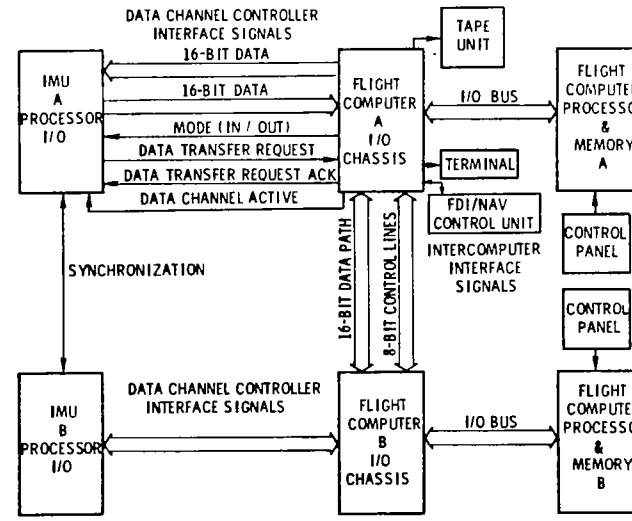


Figure 6. Block Diagram of RSDIMU Computing Hardware

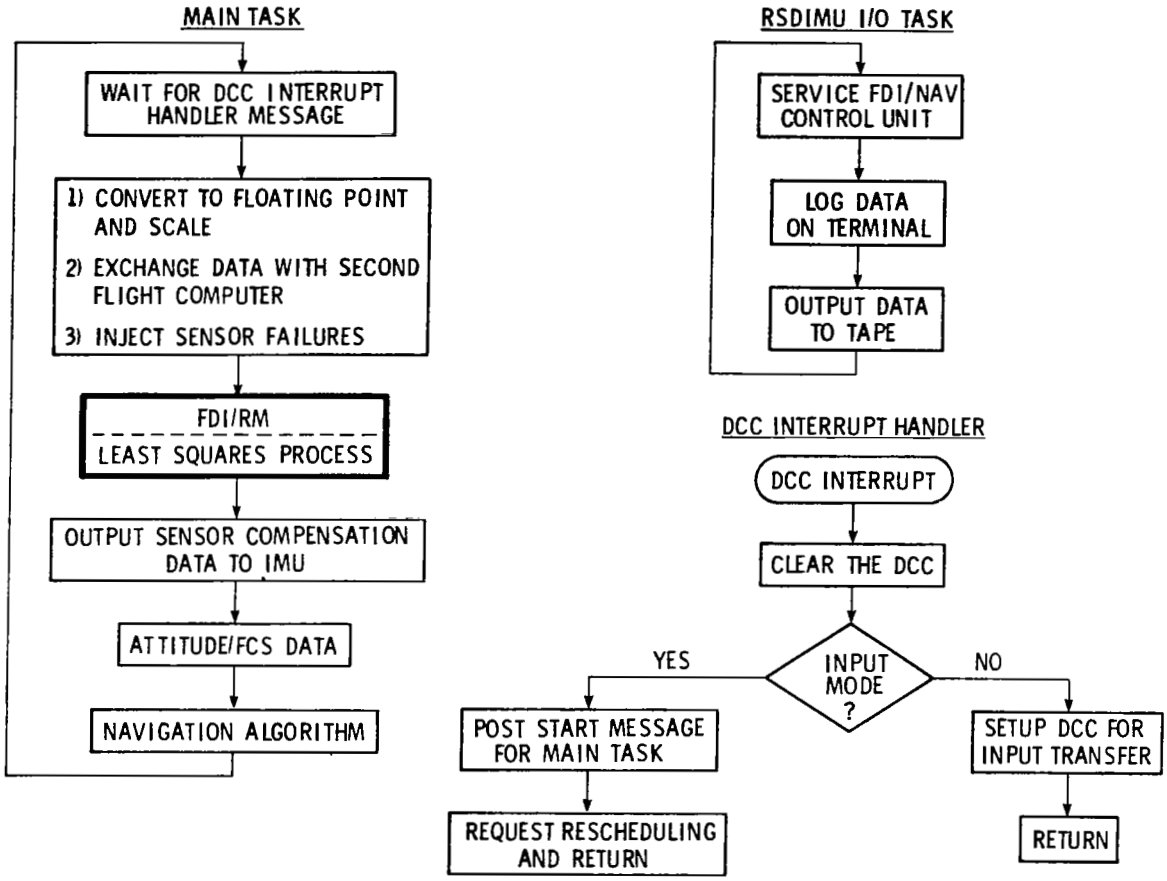


Figure 7. RSDIMU Program Flow - Navigation Mode

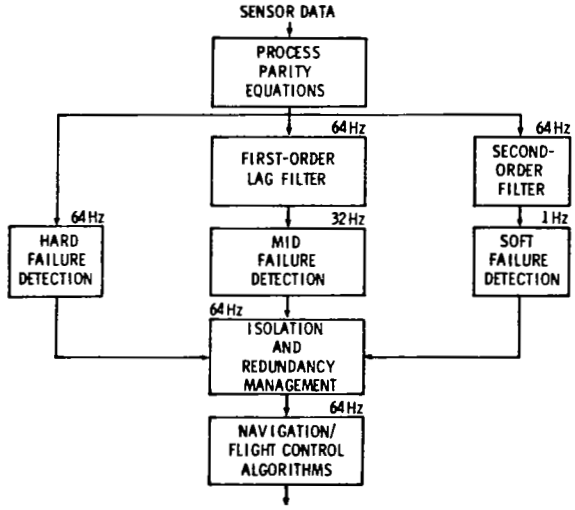


Figure 8. Failure Detection and Isolation Data Flow

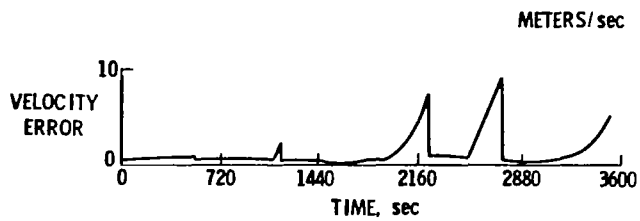


Figure 9. Velocity error and reset

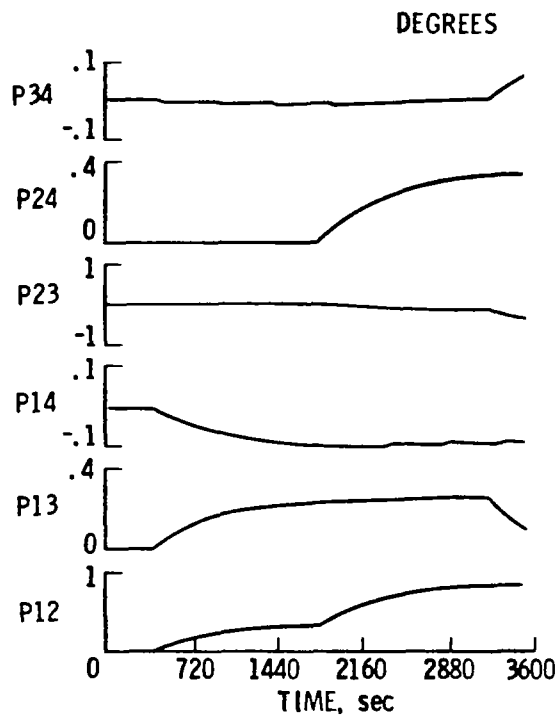


Figure 10. Gyro parity equation response to failure

ATTENDEES

James H. Bamford
Vitro Laboratories
14000 Georgia Avenue
Silver Spring, MD 20910

Major M. Barrette
Canadian Forces Air Navigation School
C. F. B. Winnipeg
Westwin, Manitoba
Canada R2R OTO

Wayne H. Bryant
NASA LaRC
MS 494
Hampton, VA 23665

Wolfgang Buechler
Comptek Research, Inc.
44 Castilian Dr.
Goleta, CA 93017

L.R. Cecchini
E-Systems, Inc., Melpar Division
7700 Arlington Blvd.
Falls Church, VA 22046

Michael J. Christofferson
E-Systems, Inc., Melpar Division
7700 Arlington Blvd.
Falls Church, VA 22046

Willie Chun
General Dynamics
P.O. Box 80847
San Diego, CA 92138

Walter K. Daku
Vitro Laboratories
14000 Georgia Avenue
Silver Spring, MD 20910

Mr. Destarac
Aerospatiale
B.P. No. 3153
Toulouse Cedex, France

James R. Doane
Sierra Research Corporation
Box 222
Buffalo, NY 14225

Charles Donaghe
Halliburton Services
P.O. Box 1431
Duncan, OK 73533

James Doty
Bendix Corporation
Guidance Systems
400 S. Beiger St.
Mishawaka, IN 46544

Albert H. Dumas
Westinghouse
P.O. Box 1897, MS923
Baltimore, MD 21203

Roy Farrow
Litton Systems (Canada) Ltd.
25 Cityview Drive
Rexdale, Ontario
Canada M9W 5A7

John B. Fisher
Vitro Laboratories
Cruise Missile Department
14000 Georgia Avenue
Silver Spring, MD 20910

Dieter Frank
AEG-Telefunken
A-2, E232
Elisabethenstr 3
79 ULM, Germany

Robert C. Garvey
E-Systems, Inc., Melpar Division
7700 Arlington Blvd.
Falls Church, VA 22046

Dean Gorby
General Dynamics
P.O. Box 80847, MS 43-5530
San Diego, CA

Joseph S. Grosson (Guest Speaker)
Executive Director for Acquisition
Naval Material Command
NAV-MAT-08B
Navy Department
Washington, D.C.

Mr. Guyot
Aerospatiale
BP No. 3153
Toulouse, Cedex, France

H.R. Helbig
Boeing Commercial Airplane Co.
P.O. Box 3707
MS 25-09
Seattle, WA 98124

Travis L. Herring
Code K105
Naval Surface Weapons Center
Dahlgren Laboratory
Dahlgren, VA 22448

Lt. Douglas Hildebrand
Scientific Analyst
USAF
4950 TESTW/FFTF
Wright-Patterson AFB, OH

Carl L. Hruby
Sierra Research
P.O. Box 222
Buffalo, NY 14225

Tom Iverson
CUBIC
9333 Balboa
San Diego, CA 92123

Joseph Jaksic
Transport Canada TAFS
Tower C, Place de Ville
Ottawa, Ontario
Canada K1A ON8

James H. King
SAI Technology Co.
4060 Sorrento Valley Blvd.
San Diego, CA 92121

Jean-Pierre Lafargue
Battelle-Institute. V.
6000 Frankfurt am Main
Am Romerhof 35
Frankfurt, West Germany 9000160

M. Langlade
SN1 Aerospatiale
BP No. 3153, 31053
Toulouse Cedex
France

Ken Lee
Kollmorgen
Route 5
North Hampton, MA

Max Lee
Lockheed
P.O. Box 551
P72-71, B310, B6
Burbank, CA 91520

General Young Soo Lee
Young O Inc. Co., Ltd.
4th Floor, Sae-Woo Building
1-499 Yoido-Dong,
Yungdungpo-Koo
Seoul, Korea

Mr. Lordemann
GTE Sylvania
P.O. Box 188
Mountain View, CA 94042

Elwood Mac Murro
U.S. Naval Underwater
Systems Center
New London, CT 06320

P.J. Manders
National Aerospace Lab.
2 Anthony Fokkerweg
1059CM
Amsterdam, The Netherlands

Orin E. Marvel
Honeywell, Inc.
1200 E. San Bernardino Road
West Covina, CA 91790

Michael McCune
Command, Control &
Communications Corp.
23670 Hawthorne Blvd.
Torrance, CA 90505

Wayne Mills
U.S. Naval Surface Weapons Center
Dahlgren Laboratory MS F24
Dahlgren, VA 22448

Jong Myung
Daejeon Machine
Depot (4-2-3)
Daejeon P.O. Box 35
Daejeon, Korea

Sharon Neelands
Singer-Librascope
833 Sonora Ave.
Glendale, CA 91201

Dennis Nickle
E-Systems, Inc., Melpar Division
7700 Arlington Blvd.
Falls Church, VA 22040

Harriet A. Nixon
Naval Ocean Systems Center
271 Catalina Blvd.
San Diego, CA 92152

Ellen Paz
Raymond Engineering
217 Smith St.
Dept. 15, Military Tape Recorders
Middletown, CT 06457

Edward Pinson
MITRE Corp.
Bedford, MA

Nancy Dull Pioli
Sierra Research Corporation
P.O. Box 222
Buffalo, NY 14225

William D. Pittman
Boeing Commercial Airplane Co.
P.O. Box 3707 Ms 25-09
Seattle, WA 98124

Mark Plummer
GTE Sylvania
Western Division
P.O. Box 188, Dept. 3100
Mountain View, CA 94042

Bernard L. Portley
U.S. Army
CORADCOM Field Office
DRDCO-FL
Ft. Leavenworth, KS 66027

Gordon P. Pratt
Goodyear Aerospace
Bldg. 26C
Litenfield Park, AZ 85340

Charles B. Probert
Vitro Laboratories
14000 Georgia Avenue
Silver Spring, MD 20910

Mr. Reau
SNI Aerospatiale
BP No. 3153, 31053
Toulouse Cedex, France

John Riney
General Systems Corporation
8611 Bells Mill Rd.
Potomac, MD 20854

Robert Shultz
Command, Control &
Communications Corp.
23670 Hawthorne Blvd.
Torrance, CA 90505

Roy L. Smith
DCA/CCTC/C332
Pentagon, Room 685
Washington, D.C. 20301

Ron Sorace
Singer-Librascope
833 Sonora Ave.
Glendale, CA 91201

Steve Spurlin
Harris Corp.
P.O. Box 37
MS 1-920
Mel Bourne, FL 32901

Darryl B. Stephison
Flight Test Instrumentation
P.O. Box 3707, M/S 25-09
Seattle, WA 98124

Steve Sultany
General Dynamics
P.O. Box 80847
San Diego, CA 92138

Carl W. Symborski
Vitro Laboratories
14000 Georgia Avenue
Silver Spring, MD 20910

William E. Tipton Jr.
USAF 3390TH TTMKM
KEESLER AFB, MS 39534
207 Coolidge Avenue
Biloxi, MS 39532

Gilliam Tucker
Comptek Research, Inc.
44 Castilian Dr.
Goleta, CA 93017

Alan Van Boven
GTE Sylvania
P.O. Box 188, Dept. 3100
Mountain View, CA 94042

David Walker
Computer Science Corp.
Dept. 652,
300 Sparkman Dr.
Huntsville, AL 35807

Larry Walker
CORADCOM Field Office
Room 24, Bldg. 52 DRDCO-FL
Ft. Leavenworth, KS 66027

Bryan Wilkins
Systems Development Laboratory
(TAFS)
Transport Canada Building
Place De Ville
Ottawa, Ontario K1A-ON8
ATTN: TANF/A

Trevor Williams
DNCO, Department of Defense
Campbell Park Offices
Canberra, Australia ACT2600

ROLM Corporation Attendees

Jim Basiji
Engineering

Steve Bowden
Sales

Rex Cardinale
Software Engineering

Frank Chang
Product Management

Denny Chrismer
International Sales

Rich Conley
Major Programs

Rich Coon
Systems Engineering

Jay Daley
Sales Support

Ed Dolinar
General Sales

Roger Fairfield
Engineering

Carol Foreman
Product Management

Dana Hendrickson
Product Management

Irv Hecker
Planning

Larry Hughes
Customer Service

Al Miller
Sales

Fran Miller
Product Management

Helmut Muehl-Kuehner
International Sales

John Myer
Product Management

Bruce Noel
Product Management

Dan O'Brien
Marketing

Alice Oldham
Systems Engineering

Dennis Paboojian
Mil-Spec Division
General Manager

Arvin Perry
Sales

Steve Phillips
Manufacturing

Dave Pidwell
Program Management

Ron Platz
Customer Service

Jim Russell
Customer Service

Don Smith
Sales

Val Smith
Sales

Ben Stanger
Technical Publications

Chuck Sternhagen
Customer Service

Larry Stovall
Program Management

John Tinnon
Software Engineering

Peter Torgrimson
Product Management

Dave Williams
Sales

Don Williams
Manufacturing

Nomi Williams
Systems Engineering

Tod Williams
Program Management

Bob Zehnder
Sales

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CP-2206		

4. Title and Subtitle	5. Report Date
RUGGEDIZED MINICOMPUTER HARDWARE AND SOFTWARE TOPICS - 1981	December 1981
	6. Performing Organization Code: 505-41-63-02
7. Author(s)	8. Performing Organization Report No.: L-14888
	10. Work Unit No.
9. Performing Organization Name and Address: NASA Langley Research Center, Hampton, VA 23665	11. Contract or Grant No.
	13. Type of Report and Period Covered: Conference Publication
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report is a compilation of conference papers presented at the Fourth ROLM MIL-SPEC Computer Users Group Conference held in San Diego, California, February 22-25, 1981. This conference provided the attendees with some insight into many novel minicomputer applications as well as providing some useful techniques for the development of error-free software. While all presentations focused on the use of a single vendor's line of minicomputers, the novel ideas described have a much wider applicability. The presentations covered both hardware and software areas in a variety of topics such as (but not limited to) the role of minicomputers in the development and/or certification of new aircraft, a minicomputer-based research tool for navigation/flight control sensor redundancy management studies, and techniques for the rapid development of error-free real-time software.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
ROLM minicomputers Airborne/spaceborne computers Real-time operation Operating systems Computer programs	Unclassified - Unlimited Subject Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	226	A11

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1981