AD-A055 591 RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 1/3
AFAL SIMULATION FACILITY/CAPABILITY MANUAL. VOLUME I. EXECUTIVE--ETC(U)
JUN 77 R A WHISNANT, W H RUEDGER, R L EARP F33615-76-C-1308
UNCLASSIFIED AFAL-TR-77-118-VOL-1 NL

1 OF 5
AD
A055 591

AFAL-TR-77-118

FOR FURTHER TRAN

AFAL SIMULATION FACILITY/CAPABILITY MANUAL.

VOLUME I.

EXECUTIVE SUMMARY AND SYSTEMS AVIONICS DIVISION.

RESEARCH TRIANGLE INSTITUTE
RESEARCH TRIANGLE PARK, N.C. 27709

(11) 30 JUN 77 (12) 411p. (16) 2003 (17) 03

(15) F33615-76-C-1308

(18) AFAL

(19) TR-77-118-VOL-1

TECHNICAL REPORT AFAL-TR-77-118
(9) FINAL REPORT FOR PERIOD JUL 76 - JUN 77
Technical

Approved for public release; distribution unlimited.

(10) Richard A. Whisnant, W. Howard Ruedger
Ronald L. Earp James Haidt

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

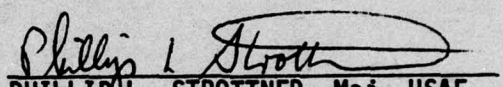
NOTICE

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

This report has been reviewed by the Information Office (IO) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

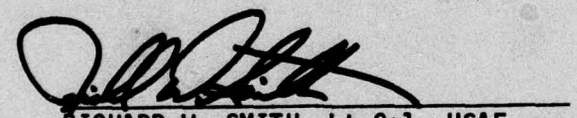
This technical report has been reviewed and is approved for publication.

THOMAS J. GREEN, Jr, 1/Lt, USAF
Project Engineer



PHILLIP L. STROTTNER, Maj, USAF
Chief, System Simulation Branch

FOR THE COMMANDER



RICHARD W. SMITH, Lt Col, USAF
Act'g Chief, System Avionics Division
Air Force Avionics Laboratory

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE/56780/16 May 1978 — 60

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFAL-TR-77-118, Volume I	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AFAL Simulation Facility/Capability Manual		5. TYPE OF REPORT & PERIOD COVERED Technical - Final Report 20 June 1976 - 30 June 1977 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard A. Whisnant, W. Howard Ruedger, Ronald L. Earp, James Haidt		8. CONTRACT OR GRANT NUMBER(s) F33615-76-C-1308
9. PERFORMING ORGANIZATION NAME AND ADDRESS Research Triangle Institute P. O. Box 12194 Research Triangle Park, North Carolina 27709		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2003-03-15
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory AFAL/AAF, Wright-Patterson AFB, Ohio 45433		12. REPORT DATE July 1977 13. NUMBER OF PAGES 391
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation	Digital Computers
Avionics	Computer-Aided Design
Digital Simulation	DECsystem-10
Avionic Simulation	PDP-11

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Air Force Avionics Laboratory (AFAL) at Wright-Patterson AFB is the focal point for development of new avionics technology for the Air Force. In order to carry out this responsibility, a significant capability to simulate physical avionics systems and components has been created by the AFAL divisions. Of prime concern is the effective use of these simulation facilities in the face of continually increasing performance requirements,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

technological advances, and rising flight-test costs.

The usual approach to satisfy requirements for increased avionics performance has been to place emphasis on the selection of the best subsystems available or on the creation of new subsystems. However, allowing subsystem performance to drive avionics system design results in inflated costs and problems in maintenance and retrofit. Subsystems that are designed for maximum performance become increasingly complex and are often incompatible unless interface requirements are considered early in the design effort. This effort requires not only a conceptual plan, but a realistic evaluation of how the coupled subsystems will interact under all critical flight conditions.

The trends toward consideration of avionics hardware from the systems' viewpoint and toward the increasing use of modularized, digital hardware put increasing demand on effective use of simulation facilities to ensure reliable, cost-effective avionics systems.

This Facility/Capability Manual for the simulation facilities of AFAL has been developed as a means for increasing the effectiveness of these important technical resources.

The primary objective of this manual is to document the total simulation capability in a manner which will serve several groups:

1. Those members of the AFAL directorate charged with planning or approval of the simulation facilities.

2. Potential users with a need to understand the general capabilities and limitations of the simulation facilities.

3. Actual users of the facilities with a need to plan simulations, document input data, conduct or coordinate simulations, and interpret results.

4. Members of the AFAL staff who are involved in updating, enlarging, or deleting simulation capabilities.

A secondary objective of this manual is to document the relationships between the various facilities, which may enhance their interaction and, thus improve the cost-effectiveness of the overall AFAL simulation capability.

The manual achieves these objectives by presenting introductory and summary material in Section I and by presenting more detailed descriptive material in subsequent sections. The contents of Section I address the Laboratory capabilities from a planning/management viewpoint by relating the Laboratory mission to present facility capability through the development of a conceptual simulation class structure. The contents of subsequent sections of this manual address specific facility/capability from a potential-user viewpoint. Both hardware and software availability are documented. The technical level of these sections is such that available capability can be determined and some insight can be gained regarding user interface.

TABLE OF CONTENTS

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

TABLE OF CONTENTS (con.)

LIST OF FIGURES

LIST OF FIGURES (con.)

LIST OF FIGURES (con.)

LIST OF TABLES

SECTION I
INTRODUCTION AND EXECUTIVE SUMMARY

The Air Force Avionics Laboratory (AFAL) at Wright-Patterson Air Force Base is the focal point for development of new avionics technology for the Air Force. In order to carry out this responsibility, a significant capability to simulate physical avionics systems and components has been created by the AFAL divisions. Of prime concern is the effective use of these simulation facilities in the face of continually increasing performance requirements, technological advances, and rising flight-test costs.

The usual approach to satisfy requirements for increased avionics performance has been to place emphasis on the selection of the best subsystems available or on the creation of new subsystems. However, allowing subsystem performance to drive avionics system design results in inflated costs and problems in maintenance and retrofit. Subsystems that are designed for maximum performance become increasingly complex and are often incompatible unless interface requirements are considered early in the design effort. This effort requires not only a conceptual plan, but a realistic evaluation of how the coupled subsystems will interact under all critical flight conditions. New technology in components for avionics systems continually suggests new system implementations, which must be explored. For example, there is a distinct trend at the present towards digital techniques in all aspects of electronics. Microprocessor technology has promoted greater utilization of software for functions previously performed by hardware. The trends toward consideration of avionics hardware from the systems' viewpoint and toward the increasing use of modularized, digital hardware put increasing demand on effective use of simulation facilities to ensure reliable, cost-effective avionics systems.

This Facility/Capability Manual for the simulation facilities of AFAL has been developed as a means for increasing the effectiveness of these important technical resources.

The primary objective of this manual is to document the total simulation capability in a manner which will serve several groups:

1. Those members of the AFAL directorate charged with planning or approval of the simulation facilities.
2. Potential users with a need to understand the general capabilities and limitations of the simulation facilities.
3. Actual users of the facilities with a need to plan simulations, document input data, conduct or coordinate simulations, and interpret results.

4. Members of the AFAL staff who are involved in updating, enlarging, or deleting simulation capabilities.

A secondary objective of this manual is to document the relationships between the various facilities, which may enhance their interaction and, thus, improve the cost-effectiveness of the overall AFAL simulation capability.

The manual achieves these objectives by presenting introductory and summary material in section I and by presenting more detailed descriptive material in subsequent sections. The contents of section I address the laboratory capabilities from a planning/management viewpoint by relating the laboratory mission to present facility capability through the development of a conceptual simulation class structure. The contents of subsequent sections of this manual address specific facility/capability from a potential-user viewpoint. Both hardware and software availability are documented. The technical level of these sections is such that available capability can be determined and some insight can be gained regarding user interface. For more detailed simulation facility capability and utilization, the reference documentation should be consulted.

Each of the divisions within AFAL has been assigned a major section heading within the manual organization. Thus, section II describes the capability/facility resident within the Systems Avionics Division; section III those within the Electronic Technology Division; section IV those within the Electronic Warfare Division; and section V those within the Reconnaissance and Weapon Delivery Division. Because of time and funding constraints under the current contract, the extent of the manual has been limited to the simulation facilities within the Systems Avionics Division. It is anticipated that the manual will be expanded in the future to include the remaining AFAL Divisions.

1.1 AIR FORCE AVIONICS LABORATORY: MISSION AND ORGANIZATION

The Air Force Avionics Laboratory is the principal development organization within Air Force Systems Command (AFSC) for new avionics technology. The primary responsibility of AFAL is to provide the USAF with the products and expertise required for the acquisition of the best possible avionics systems. To meet this responsibility, AFAL maintains a base of avionics technology and develops and demonstrates cost-effective avionics systems and subsystems to improve operational capabilities in navigation, communications, electronic warfare, surveillance, reconnaissance, and weapon delivery. As required, the laboratory provides technical assistance in the systems-acquisition process to all elements of AFSC, and to all USAF elements in the operation and modification of avionics equipment in the inventory.

The laboratory is organized into four major technology groups, as shown in the organization chart in Figure 1.1-1. System and device development is conducted in the Systems Avionics, Reconnaissance and Weapon Delivery, Electronic Technology, and Electronic Warfare Divisions. The mission of each of these is discussed briefly in the following paragraphs.

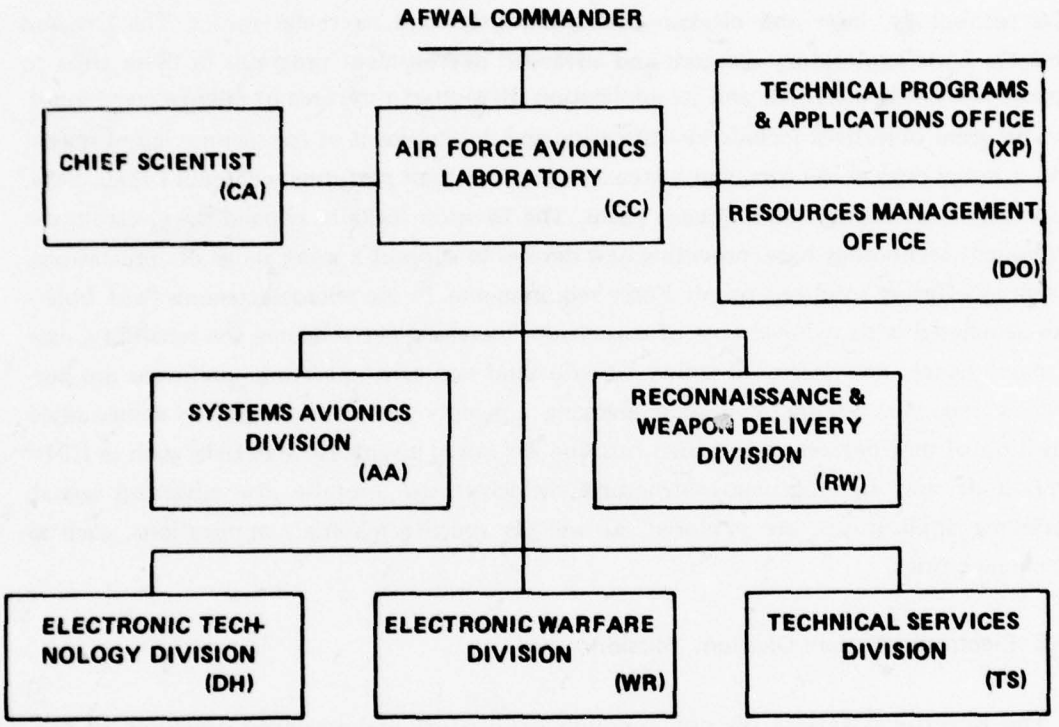


Figure 1.1-1. AFAL organization.

1.1.1. Systems Avionics Division: Mission

The Systems Avionics Division develops concepts and methodology for the architecture of advanced avionics systems. Research and exploratory development are conducted in the areas of information handling, processing, and transfer; display and control; subsystem design and optimization; subsystem and functional integration; and electromagnetic transmission links. Programs within the Division develop and demonstrate advanced concepts of avionics subsystems integration, automation, and information processing, display, control, and transfer. In-house capability is being developed to define, demonstrate, and test digital

avionics and to provide the hot-bench facility and expertise necessary to achieve this capability. The Digital Avionics Information System (DAIS) is an example of such a program.

1.1.2 Electronic Technology Division: Mission

The Electronic Technology Division maintains centers of excellence in radar and microwave technology, laser and electro-optic technology, and microelectronics. The Division conducts basic exploratory research and advanced development programs in these areas to support the needs of AFAL and its application divisions. In the area of microwave technology, program objectives include identification and development of the technology of microwave avionics devices, sensors, and systems to improve their performance, reduce their costs, and evaluate alternative development paths. The Division seeks to expand the electro-optic component technology base, providing new devices to support a wider range of applications, as well as offering solutions to Air Force requirements. In the microelectronics field, objectives associated with reduced cost of ownership, increased performance and reliability, ease of maintenance, and increased ability to withstand stringent operating conditions are pursued by recognizing and exploiting emerging concepts and technologies to assure rapid transition of new devices and circuits into the Air Force inventory. Materials, such as III-IV compounds and heterogeneous structures, which show promise for advanced signal-processing applications, are explored, as well as specific materials applications, such as bubble memories.

1.1.3 Electronic Warfare Division: Mission

The Electronic Warfare Division conducts exploratory and advanced development programs in the technical domain of electromagnetic warfare. The Division participates in advanced planning to provide effective guidance for the Division mission and to provide timely transition of exploratory and advanced development programs into effective military hardware. In carrying out its mission, the Division maintains electronic simulators, such as the Electronic Defense Evaluator (EDE) and the Dynamic Electromagnetic Environment Simulator (DEES). Technical areas of interest include, among others: analysis of threats, projecting scenarios, modeling the effects of electronic warfare on penetration effectiveness, and performing tradeoff evaluations; performance of electronic warfare technique analysis to generate EW effectiveness data; development and demonstration of the military worth of countermeasures for defense of aerospace vehicles against threats utilizing optical or electro-optical guidance or fire control systems; and development of advanced techniques, tactics, and equipments for manned aircraft to penetrate or enter a hostile air environment.

1.1.4 Reconnaissance and Weapon Delivery Division: Mission

The Reconnaissance and Weapon Delivery Division conducts exploratory and advanced development programs to demonstrate improved aerospaceborne reconnaissance, navigation and weapon delivery capabilities for present and future Air Force tactical and strategic weapon systems. The Division conducts studies and analyses of potential concepts, subsystem requirements, aerospaceborne reconnaissance, navigation, target acquisition, fire control and weapon delivery avionics subsystems, and provides promising completed exploratory efforts for incorporation into those subsystems. Another Division function is identification of areas of technology in its area of interest, which require development by other AFAL organizations. Also, the Division maintains a group for dynamic mobile evaluation of software for aerospace inertial/reference subsystems and a group for dynamic and environmental evaluation of avionic sensors, subsystems, and systems.

1.2 SIMULATION CLASS STRUCTURE

In the same sense that avionic hardware development tends to proceed within the *narrow confines of the functions* and performance of specific subsystems—for example, a radar or voice communication radio—simulation capabilities also tend to be developed for aid in design of certain specific subfunctions of avionics. In recent years, however, the opportunity for and desirability of the integration of avionics subsystems into a functional hardware system have presented the necessity to examine the tradeoffs required by the system-integration process.

While the simulation capabilities that are a vital part of subsystem design are as necessary as ever, simulation of systems at a higher aggregate level become equally important. In fact, a hierarchy of simulation capabilities becomes desirable and necessary to insure that overall system performance meets mission requirements and that each subsystem satisfies its own subrequirements. When simulation is viewed from the perspective of assessing the total avionic function as an integrated system, it is convenient to structure it to parallel a top-down-system design procedure that results in maximizing total system performance.

This section presents a conceptual simulation architecture encompassing several levels of simulation support related to the avionics design and integration process. The rationale for this architecture was originally developed for the Avionic System Analysis and Integration Laboratory (AVSAIL) facility (Section 1.3.1) in the Systems Avionics Division. However, the architecture is designed to encompass a total system approach to simulation of avionics, rather than a subsystem approach, so that it is a useful framework for relating the function

of simulation facilities/capabilities in other AFAL Divisions as well. As may become apparent, such a class structure must occasionally be warped slightly to fit an individual case, but in general it provides a useful framework for both laboratory management and user. By categorically locating given simulation capability within such an architecture, it makes it convenient for facility management to plan for controlled expansion of capability and also for the potential user to select and review only those capabilities which are relevant to the simulation detail required for his particular system design task. These categories are listed below in Table 1.2-1. Table 1.2-2 shows applications, outputs, and examples of the various levels as configured at the Avionics Laboratory.

TABLE 1.2-1. SIMULATION LEVELS

Level I – System functional	Level V – Real-time dynamic
Level II – Discrete event	Level VI – Real-time sensor signal level
Level III – Scientific	Level VII – Special purpose hybrid
Level IV – Interpretive computer	

These various simulation categories may operate as separate capabilities for designers whose problems fit a particular level, or where evaluation is needed of a specific proposed configuration. The designer may also use the categories as a continuous system of levels starting at the broad view and proceeding to finer levels of detail. Iteration may occur among any of the levels at any point in the simulations.

At the broadest level of Systems Simulations, the designer looks at functions and obtains general, variable, probabilistic outputs. At Discrete Event Simulation, the exterior boundaries of the proposed system become more specific. At this level the designer gains an understanding of what the hardware in the proposed system would look like and how the subsystems would interconnect and communicate. In Scientific Simulation, the software is added in the form of algorithms, and the entire system is placed inside an external environment model group so that flight conditions may be reproduced. In Interpretive Computer Simulation, the subsystem algorithms would be changed to actual flight codes. In Real-Time Dynamic Simulation, an actual subsystem interconnects with the simulation and replaces a model. Most of the proposed avionics system would continue to be in modeled form since this test concerns the integration and interface of a single subsystem. In Real-Sensor Signal Level Simulation, groups of functional hardware would replace groups of models for a complete system integration test. The external environment remains modeled and interfaced, but actual sensors may also be stimulated to carry the environmental simulation to further details.

TABLE 1.2-2. SIMULATION CLASSES

Classes	Applications	Outputs	Examples
I. System Functional	System Analysis System Design System Requirements Definition System Level Trade Studies Parametric Analyses	Performance Parameters/Limits Sensitivity Thresholds Criteria Requirements Mission Scenarios System Reliability Estimates	AEP Cost Models
II. Discrete Event	Computer System Capabilities Throughput Analyses I/O Requirements Loading Information Transfer Requirements Loading Benchmark Testing System Time Line Analysis	Processing Requirements CPU Sizing Bus Loading System Level Timing	Simnuc DPM GASP IV
III. Scientific	Navigation/Sensor/Flight Dynamics Interactions Algorithm Verification System Performance Evaluation Subsystem Design	Closed Loop Navigation Operation Ideal Weapon Def. Evaluation Statistical Data Collection Processor Characteristics/ Design Mux System Design System Software Development	SDVS AVSIM
IV. Interpretive Computer	Detailed Timing Evaluation Fine Grain System Interaction Flight Computer Code Evaluation Detailed Design of Processor/Mux Subsystems	Detailed Timing of Command/Discretes Detailed "Debug" Aid for Flight Code Preliminary Hardware Timing Evaluations	SDVS (ICS) Processor Architecture (ISP)
V. Real-Time Dynamic	Man-In-Loop Evaluations (cockpit only) OFP & Flight Computer Interaction Functional Avionic Representation	Mission Scenario Evaluation Sensor Modeling Verification Control/Display Evaluation Phase IV & V of OFP OFP/Flight Computer Integration	AVSIM

TABLE 1.2-2. SIMULATION CLASSES (con.)

Classes	Applications	Outputs	Examples
VI. Real-Time Sensor Signal Level	System Integration Avionic Sensor Interface Verification Integrated System Testing Dynamic Simulation Testing of Sensors	End-to-End Signal Flow Testing Hardware/Software Dynamic Interaction Completely Instrumented Integrated System Tests Phase IV & V of OFP	DAIS
VII. Special Purpose Hybrid	A-J Signal Structures Les 8/9 Support GPS Support RPU Data Link Design	Verified A-J Signal Structures Performance vs. Channel Characteristics	CSEL

These simulation levels are discussed individually and in greater detail in the following paragraphs.

1.2.1 Level I, System Functional Simulations

System simulations are a starting point for defining, in system terms, the avionic functions needed. In a computer simulation of the functional capabilities of an avionics system, analyses of the many options available provide the systems designer with a means of determining the optimum avionics configuration for a selected mission. Mission requirements are first analyzed and allocated to specific functions and modes of operation for each phase of each postulated mission. Appropriate hardware configurations are then defined, and the operating characteristics specified to create a system simulation model. A computer analysis is then made to evaluate the performance of each configuration relative to various mission requirements. Overall mission performance, as well as candidate system cost-effectiveness, are elements of such an analysis.

Typically, the system designer defines the characteristics applicable to the desired aircraft type and avionics system configuration; then he selects the mathematical models necessary to accomplish the simulation from AVSAIL's library of simulation software. He may then successively modify the configuration by deleting or adding various subsystem models.

Comparision of the simulation runs and the system performance requirements will suggest tradeoffs, which could result in modifying the initial system design. These modified configurations may then be subjected to additional simulation analysis.

The system designer may vary numerous model characteristics and determine the effect of the changes on overall system performance. He may utilize various configurations to examine the operation of each avionics subsystem and gain an understanding of the functional performance necessary to insure that the system is responsive to primary mission requirements. Thus, with AVSAIL's capabilities, the system designer can examine the effects of varying the system's configuration in a logical and orderly manner, prior to construction of hardware prototypes. In general, this level of simulation explores and compares different modes and options for implementing the mission functions. Hundreds of simulated missions can be examined in a few minutes. Once mission functions have been established, further refinement of the system design may be accomplished with more detailed levels of simulation.

1.2.2. Level II, Discrete Event Simulations

Discrete event simulations are used to investigate the functional and physical configurations of a proposed avionics system and provide analysis of information-processing loads and transfer requirements. The designer is able to evaluate and analyze time sequences and the effects of system degradation. He looks at the proposed system configuration, and he can make comparisons of various configurations.

Given a specific configuration of avionics subsystems and the detailed performance requirements of each, the simulation defines the capability required of the airborne processor and determines the flow of information between each of the subsystems. An analysis of the amount of information flow at any given point in time provides data on gross system timing and enables the designer to reallocate functions as necessary.

The simulation also permits examination of the information-processing cycle for the avionics system by relating each operation to airborne processor time for comparision with established time boundaries. For each function, the designer can determine total time used, including processing time required or processor speed required for desired performance. Accuracy requirements may also be analyzed.

Various methods of interconnecting subsystems can be examined. The simulation permits examination of trial subsystem interconnections to determine the information flow

timing. The designer thus identifies peak loading conditions and excessive information delays.

The simulation also establishes boundaries on the amount of information flow between any subsystem and any processor or termination point at any given time. This results in the questions: What should be allocated to an airborne central processor? What processing should be broken up and allocated to local points in the system?

If, after trying partitioning alternatives, the simulation result calls for impractical airborne processor capacity or speed, then the proposed avionics system may not be a valid solution for the mission in terms of current hardware technology. However, this type of result would suggest the need to iterate with the system functional simulation, trying other modes and configurations. Feedback and interaction among various types of simulation is a conventional usage of AVSAIL in design and analysis.

The proposed partitioning may or may not be satisfactory; if not, modification or reconfiguration is necessary. The designer continues to experiment with various ways in which the system can be interconnected. When he has modeled a workable system, he has defined the hardware in terms of operating characteristics, hardware interconnection, and information flow. He has also examined information flow rates and has established general boundaries on the amount and speed of airborne processing required. His next step is to model in detail all the elements of the system.

1.2.3 Level III, Scientific Simulations

Scientific Simulations involve the use of detailed models for both hardware and software in a proposed avionics system. This level provides an opportunity for comprehensive evaluation of the proposed system's operating logic, performance, and timing, and permits examination of tradeoffs between hardware and software. The physical partitioning of the proposed system is validated, and hardware specifications and software algorithms are developed.

The detailed mathematical models of specific subsystems are constructed to include functional performance and to accept input and produce output signals. For example, a chosen model of an inertial subsystem would produce signals, which would correspond to the velocities it should sense over a particular period of time during a simulated mission.

The mathematical models are combined with a detailed information-transfer scheme and with trial software models; the entire simulation is also embedded within environmental

models. The overall environmental simulation includes validated models of airframe, rotating earth and atmosphere so that the system under test can receive realistic signals and produce results that can be accurately analyzed for functional performance.

Aside from a close look at hardware elements and partitioning, software development for the proposed avionics system is begun in Scientific Simulation by providing an operating environment to try algorithms in dynamic situations. The designer can then evaluate the performance of a particular navigation algorithm under given conditions.

Since the system elements are modeled in detail and accept realistic inputs, the simulation also provides accurate outputs in the form of actual signal levels. After suitable algorithms are tested and potential hardware/software tradeoffs are identified and chosen, detailed specifications are generated for both hardware and software.

1.2.4 Level IV, Interpretive Computer Simulations

Interpretive Computer Simulations are similar to Scientific Simulations, but software analysis is performed at the airborne processor instruction level for detailed flight-program development and an examination of fine-grain hardware and software interactions. The timing of all airborne-processing functions is recorded for analysis.

The Scientific Simulations examine various airborne software algorithms operating with detailed models of the hardware and the environment. At the Interpretive Computer Simulation level, the simulations examine complete airborne software coding, operating with similarly detailed models of hardware and environment. A compiler in the simulation computer generates the machine code so that the model of the airborne processor can operate with bit-by-bit simulation of each machine-level instruction. Feedback with other types of AVSAIL simulation may be required to refine the software coding.

Since the analysis is very detailed, the simulation is focused at critical processing points in the mission, examining short segments of airborne processor operation. Several hours of simulation are typically required to evaluate a few minutes of flight time.

Actual execution of flight code instructions provides a thorough analytical method for evaluation of software accuracy and identification of timing problems. The most common objective of this simulation is to support the development and validation of flight code.

1.2.5 Level V, Real-Time Dynamic Simulations

The primary purpose of Real-Time Dynamic Simulation is to test actual hardware operating in real time with accurate, detailed simulation models of the proposed avionics system and the environment.

Any actual avionics subsystem may be exercised to examine its real-time interactions with the total avionics system. This level of simulation is used to support integration of the particular subsystem with the proposed avionics system. For example, an airborne processor, loaded with the flight code it will run, can be driven by the simulation. Inputs would be provided to the airborne processor hardware by the simulation models. The models would also accept outputs from the hardware under test and interact realistically with the system.

The environmental simulation provides the hardware under test with a realistic operating environment, including all flight information references. A comprehensive example is the operation of a cockpit with controls and displays hardware, either to examine the hardware interfaces and interactions with the rest of the avionics system or for a man-in-the-loop study. The simulation will accept the control inputs and drive the displays according to model outputs, including the environment simulation. A video scanning and mixing capability provides realistic display background synchronized with the modeled airframe and environment. The sampling of controls and the driving of displays occur in real time at the cycle time of the proposed avionics system.

Except for such questions as the effect of dynamic maneuvering loads, Real-Time Dynamic Simulation can produce avionics system debugging results previously obtainable only through expensive and time-consuming flight test. For this level of simulation, actual hardware replaces one or more simulation models for validation of one subsystem at a time.

1.2.6 Level VI, Real-Time Sensor Signal Level Simulations

Real-Time Sensor Signal Level Simulations are the basic tool for total system integration. Groups of simulation models such as those used in Real-Time Dynamic Simulation would be replaced by multiple functional groups of actual hardware run together in real time for an integrated study. Whereas Real-Time Dynamic Simulations concentrate on a single item or a single closely related group of hardware, this integrated simulation level exercises and examines complete functional sets of hardware subsystems. For example, all hardware which will operate on information transfer and processing may be tested together.

A complete external environment is simulated to the avionics system under test, and the AVSAIL simulation computer is capable of providing appropriate stimulation to actual sensors so that overall performance of the system may be evaluated dynamically and realistically. Where it is not practical to present a computer-generated signal to an actual sensor (for example, where aircraft motion must be detected), the sensor output is modeled to conform with the missions and mission environments being evaluated. Modeled signals are injected as early in the system test as is practical.

Complete groupings of hardware are operated in a simulated real-time mission to study integrated performance, to verify that all hardware interfaces operate properly, and to validate the system software under simulated flight conditions. Virtually all mission modes can be examined using validated models of earth, atmosphere, and airframe.

AVSAIL complete system integration tests complement flight testing, particularly in software validation and hardware debugging. These simulations can replace such basic checkout procedures previously accomplished only in flight testing; subsequent flight tests can concentrate on validating dynamic performance.

1.2.7 Level VII, Special Purpose Hybrid

While not necessarily a "level" per se, the capability to perform special purpose simulations is a necessary requirement in systems design. Under the Special Purpose category, the Avionics Laboratory has developed the Communications Systems Evaluation Laboratory (CSEL).

CSEL has been developed to assist the U.S. Air Force in the analysis, synthesis, and modeling of its communications and data links, and to provide a cost-effective means for dynamic evaluation and comparison of advanced techniques and systems.

Current Air Force communications links between aircraft—both direct and via satellite—operate in the ultra high frequency (UHF) or super high frequency (SHF) radio bands. As new user requirements evolve, new communications systems and data links, such as the Lincoln Laboratories' LES 8/9 satellites, are being designed to handle them. However, new systems and innovative equipment are, in themselves, not enough to handle the ever-increasing user requirements; there is a corresponding need for change in such related areas as frequency bands of operation, signal structures, and modulation techniques. The CSEL, by providing the proper computer hardware/software mix, offers a dynamic evaluation tool that will provide the capability to observe and evaluate the performance of such advanced communications and data systems.

1.2.8 Digital Avionic Information System (DAIS)

The various levels of simulation resident within AVSAIL allow the user not only to select the appropriate degree of sophistication to satisfy his application, but more powerfully, to assemble various levels in order to broaden interactively the available simulation capability. Additionally, the user may choose to sequentially select various simulation levels as he moves through the various phases of system design. In the following discussion, the DAIS project is presented as an example of the manner in which AVSAIL can support the various programs resident within AFAL by virtue of this simulation level structure.

The purpose of the DAIS project is to demonstrate a coherent solution to the problem of proliferation and nonstandardization of aircraft avionics, to develop and test in a hot-bench configuration (known as the Integrated Test Bed) the DAIS concept, and to permit the Air Force to assume the initiative in the specification of avionics configurations for future Air Force weapon systems acquisitions. The DAIS design approach reflects a total system concept that is functionally oriented rather than hardware oriented.

The heart of the DAIS system is the redundant time division multiplex data bus shown in Figure 1.2.8-1. This bus allows information from the aircraft subsystems (e.g., avionics units, stores management, power control) interfaced by remote terminals (RT) to be communicated along the bus and to a set of shared DAIS processors through Bus Control Interface Units (BCIU) in the processors. Mission software, developed through simulation with the Software Design and Verification System (SDVS) in non-real-time interaction (Levels II and III) with aircraft and environmental models, can be exercised in real time in the ITB facility. For example, a pilot flying a simulated cockpit views a simulated, computer-generated scene and interacts with displays in the cockpit generated by DAIS mission software. The aircraft external environment and flight dynamics are simulated by models executed by the host computer in a Class II simulation. During such a simulated flight, the mission software/processor performance is monitored by the Super Control and Display Units (SCADU), while the bus performance is monitored by the Bus Monitor Unit (BMU). The results of the Level V simulation can then be compared with those predicted by earlier non-real-time simulations at Levels I to IV. System performance is, thus, verified in the laboratory instead of in the field. Many of the simulations utilized in the DAIS ITB have application to other avionics system developments and are described in Section 2.0 of this manual.

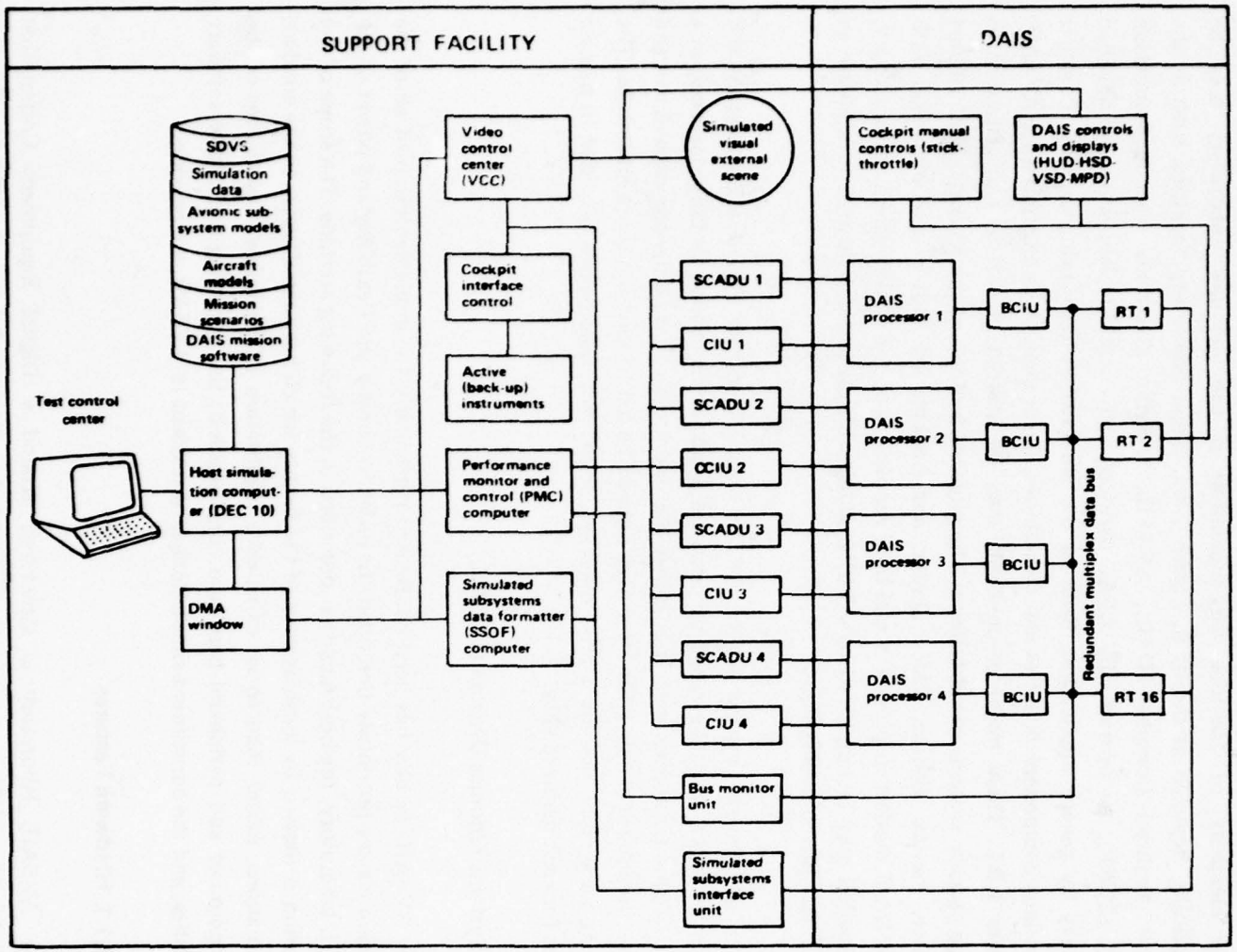


Figure 1.2.8-1. Simplified block diagram of DAIS ITB facility.

1.3 AFAL FACILITIES

The hierarchy of facilities for simulation at the Avionics Laboratory may be schematically depicted as shown in Figure 1.3-1. Major laboratory facilities, such as the Dynamic Analyzer Complex (DAC), AVSAIL, and the Electronic Warfare Simulation Facility (EWSF), are operated by AFAL divisions and constitute laboratories subdivided essentially by generic application. Each of the laboratories has varying degrees of further division into component facilities and capabilities, as for example is illustrated for AVSAIL in Figure 1.3-1. These major components may be utilized in various configurations to simulate specific systems or subsystems. The physical facilities of the AVSAIL DEC-10 host computer, Picture System, Video Center, and Cockpit are shown in conjunction with AVSIM DAIS models to provide a real-time simulation of the DAIS system. Alternatively, other models may be employed by AVSIM, along with dedicated hardware to simulate an F-16 aircraft fire control system.

The facilities and capabilities of the Avionics Laboratory are of a complexity and versatility so great that the range of potential applications cannot be fully described in a manual of this limited extent. It is rather the intent here to describe the capabilities and facilities themselves so that the user may himself be led to envision the applications. The following paragraphs present a brief overview, with additional details provided in manual sections for each major facility.

1.3.1 Systems Avionics Division (AVSAIL)

The AVSAIL facility has been configured particularly for implementation of all of the simulation classes previously described. In order to convey the flexibility and power of the AVSAIL laboratory, the host facility is described in the following sections. The scope of the description is limited to those aspects of the facility which have significance to the conduct of simulations, rather than to an exhaustive exploration of capability. Discussions of the basic computer and peripheral hardware configuration, the operating and utility software capabilities, and the constituent simulations now resident in AVSAIL are provided.

1.3.1.1 Hardware Features

The AVSAIL laboratory is structured around a Digital Equipment Corporation DECsystem-10 mainframe computer. As depicted by Figure 1.3.1.1-1, the DEC-10 has a bus architecture, which provides both a memory bus for processor-memory and direct memory access communications and an input/output bus for processor-peripheral communications.

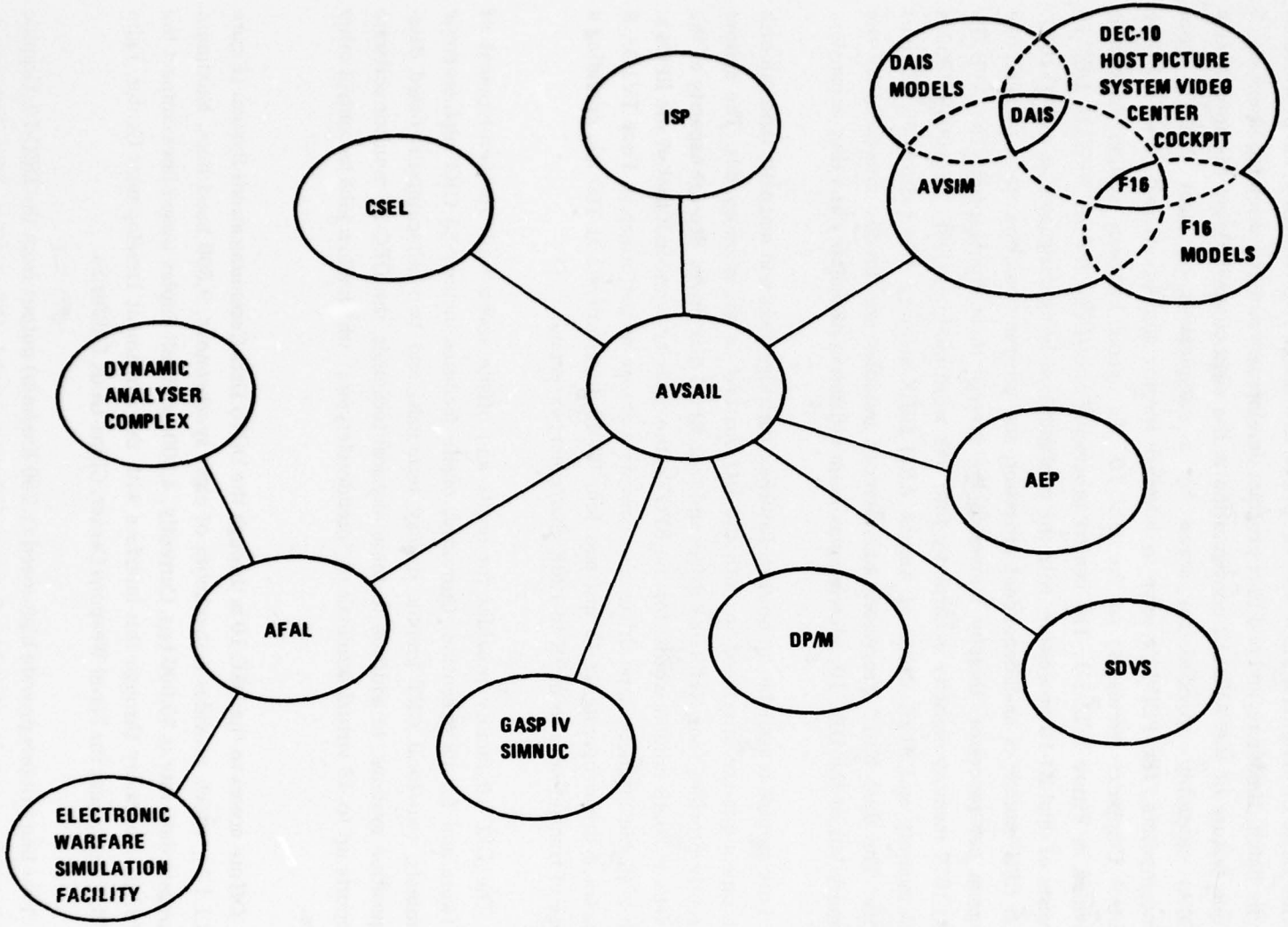


Figure 1.3-1. AFAL simulation facility organization AFAL/AVSAIL/AVSIM.

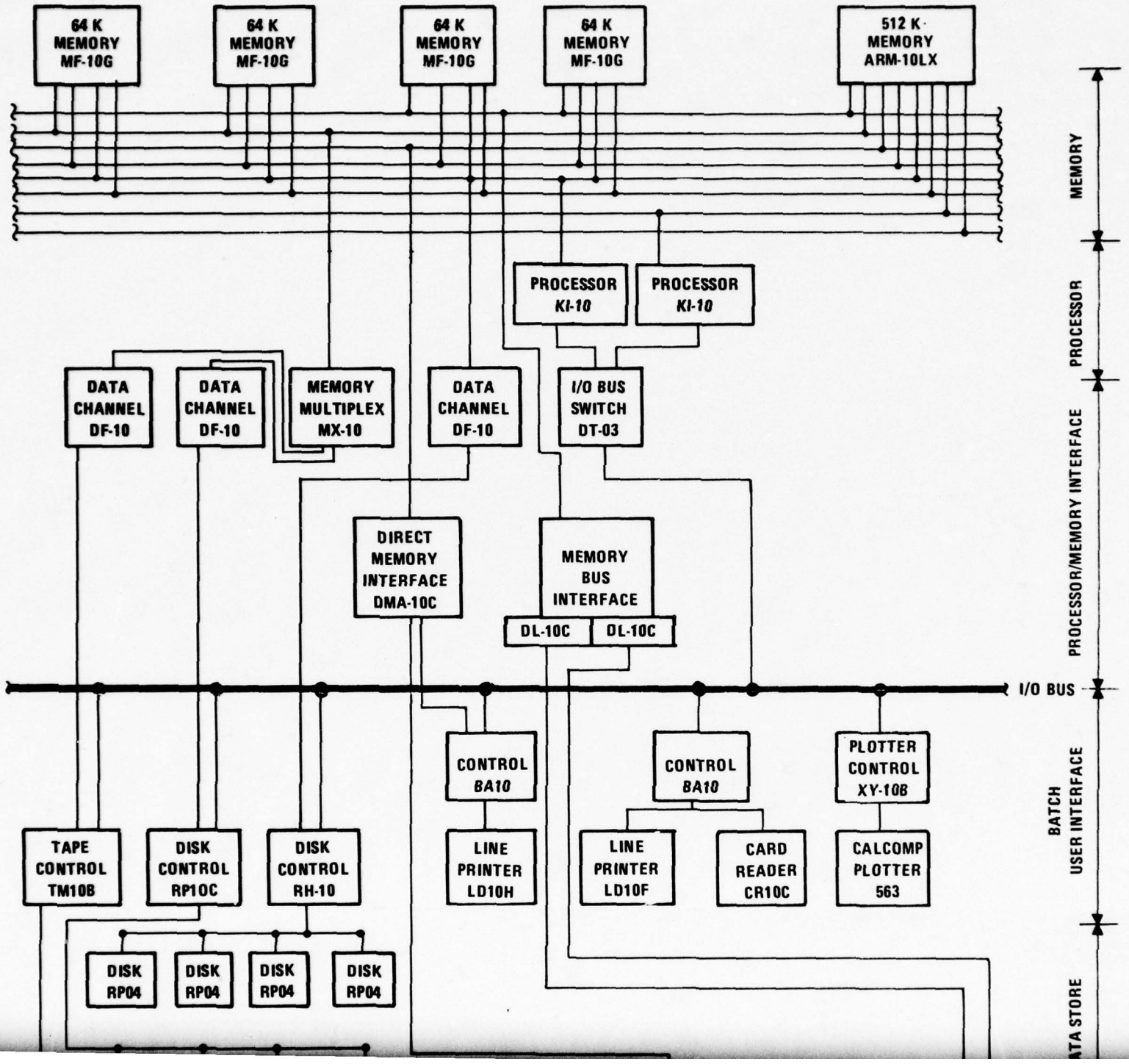
A wide range of peripheral input/output devices, as described subsequently, are provided to handle batch, timeshare, and real-time program development and execution requirements. A unique feature of the AVSAIL configuration is the eight-channel Direct Memory Access (DMA) capability provided for access by a complement of eight PDP-11 series minicomputers. The PDP-11's serve to interface specific simulations (e.g., the F-16 Fire Control Computer simulator) to the DEC-10. An overall hardware system diagram is provided in Figure 1.3.1.1-1. The central processing unit (CPU) for the AVSAIL DEC-10 consists of dual KI-10 processors, with the configuration being designated as a DEC-1077. Each CPU module is an independent processor, and programs can operate in parallel, one program per processor, thereby increasing the average throughput speed. The AVSAIL DEC-1077 memory capacity is currently four 64k word modules (DEC MF-10G) of 950ns core memory and 512K words of Ampex ARM 10LX memory. Word size is 36 bits plus parity. The dual KI-10 processors and memory modules are directly interfaced by the memory bus of the DEC-10, allowing maximum utilization of memory and easy expansion.

Bulk storage is available on both multisurface cartridge disks and magnetic tape. All disk and tape units are interfaced to both the I/O bus and to the memory bus. The system currently provides four RP03 disk drives and four RPO4 disk drives. Storage capacity of the RPO4's is 20.48 million words; for the RPO3's, the capacity is one-half that of the RPO4's. The available magnetic tape drives also provide a choice of performance. Four TV1OA-E nine-track drives operate at 45 ips, and four TV40 drives operate at 150 ips, providing a range of transfer rates from 9K to 120K characters per second.

The DEC-10 facility provides for onsite and offsite access both for development of software and for its execution. Currently, onsite facilities include 25 CRT alphanumeric terminals, two local CRT graphic display terminals, and two minicomputer-based data-acquisition systems. In addition to these physical terminals, the DEC-10 monitor software supports up to 48 virtual terminals or "pseudoteletypes,' which allow jobs to control other jobs.

Offsite access to the DEC-10 is through the DC75 Data Communications System (Figure 1.3.1.1-1), which provides a maximum of eight synchronous, 9,600 baud lines. Maximum aggregate data rate is 30,000 bps. Currently, 4,800-baud full-duplex leased lines connect the AVSAIL laboratory through this interface with the Armament Development Center, Eglin AFB, Florida, and the Naval Weapons Center, China Lake, California.

Two line printers provide high-speed (1,250 line/min) output from the DEC-10. Graphic hard-copy output is available from a *Calcomp Model* 563 plotter. This high-speed,


64 K MEMORY MF-10G
64 K MEMORY MF-10G
64 K MEMORY MF-10G
64 K MEMORY MF-10G
512 K MEMORY ARM-10LX
PROCESSOR KI-10
PROCESSOR KI-10
DATA CHANNEL DF-10
DATA CHANNEL DF-10
MEMORY MULTIPLEX MX-10
DATA CHANNEL DF-10
I/O BUS SWITCH DT-03
DIRECT MEMORY INTERFACE DMA-10C
MEMORY BUS INTERFACE
DL-10C
DL-10C
I/O BUS
CONTROL BA10
CONTROL BA10
PLOTTER CONTROL XY-10B
TAPE CONTROL TM10B
DISK CONTROL RP10C
DISK CONTROL RH-10
LINE PRINTER LD10H
LINE PRINTER LD10F
CARD READER CR10C
CALCOMP PLOTTER 563
DISK RP04
DISK RP04
DISK RP04
DISK RP04
MEMORY
PROCESSOR
PROCESSOR/MEMORY INTERFACE
BATCH USER INTERFACE
ATA STORE

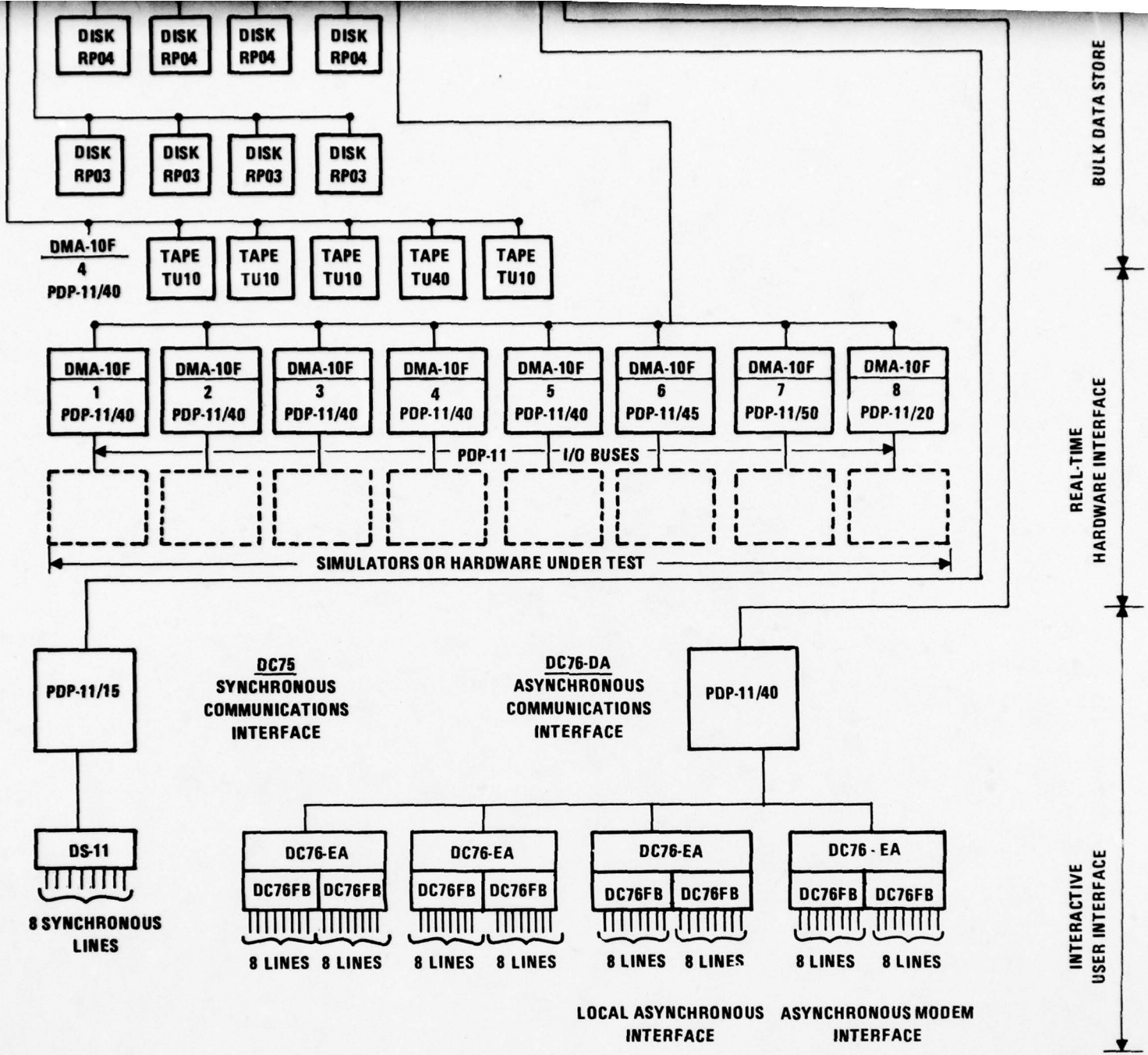


FIGURE 1.3.1.1-1. DEC-10 SYSTEM HOST SIMULATION PROCESSOR HARDWARE.

drum-type, pen and ink plotter uses 31-inch paper and operates at up to 300 steps per second. Card input to the system is available through a 1,200-card-per-minute reader.

1.3.1.2 Software Features

The wide variety of computing requirements demanded by the several classes of simulations carried out within AVSAIL are satisfied by the flexibility and scope of the DEC-10 software package. This software package provides for the concurrent operations of timesharing, multistream batch, real-time, and remote communications. These multifunction capabilities allow multiple users, both at AFAL and at remote locations, to perform all of the tasks necessary to create new simulations, modify existing simulations, and run those simulations as if they were individual users. The system allows a maximum of 48 users.

From the user's viewpoint, the DEC-10 may be thought of in terms of (1) input device and software which he has written or which act on his software, as in Figure 1.3.1.2-1; (2) the operating system software, which controls system resources; and (3) the system hardware which was previously described.

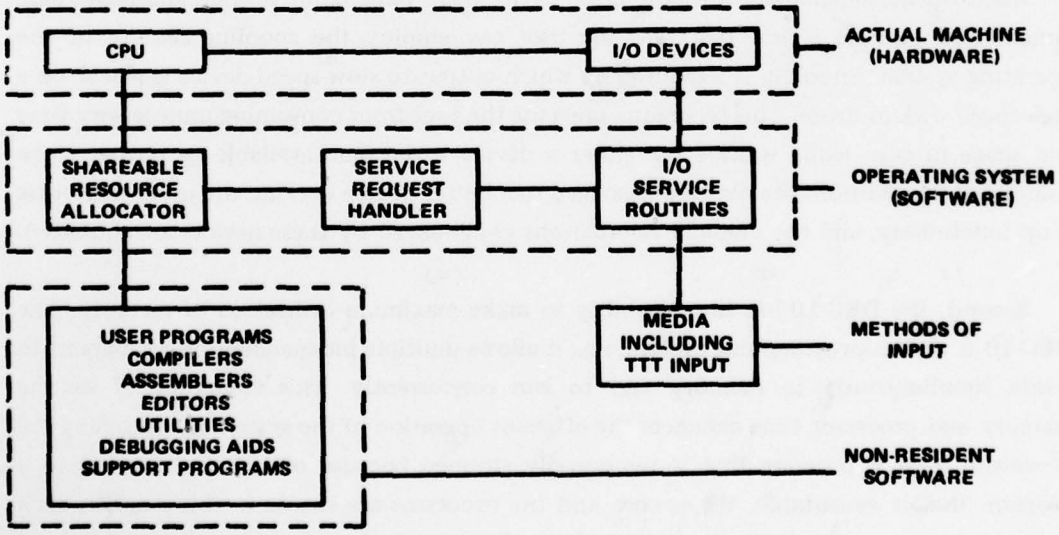


Figure 1.3.1.2-1. DECsystem-10, user's view.

The DEC-10 has several capabilities, which increase the utilization of system resources in a multiuser environment. First, the timesharing capability allows resources to be shared among users. Users are not restricted to a small set of system resources, but instead are provided with the full variety of facilities. By means of his terminal, the user has online access to most of the system's features. This online access is available through the operating system command control language, which is the means by which the timesharing user communicates with the system.

Through the command language, the user controls the running of a task, or job, to achieve the desired results: create, edit, and delete files; start, suspend, and terminate a job; compile, execute, and debug a program. In addition, since multiprogramming batch software accepts the same command language as the timesharing software, any user can enter a program into the batch run queue. Thus, any timesharing terminal can act as a remote job-entry terminal.

With the command language, the user can also request assignment of any peripheral device (magnetic tape, DECtape, and private disk pack) for exclusive use. When the request for assignment is received, the operating system verifies that the device is available to this user, and the user is granted its private use until he relinquishes it. In this way, the user can also have complete control of devices such as card readers and punches, paper-tape readers and punches, and line printers.

When private assignment of a slow-speed device (card punch, line printer, and paper-tape punch, and plotter) is not required, the user can employ the spooling feature of the operating system. Spooling is a method by which output to slow-speed device is placed on a high-speed disk or drum. This technique prevents the user from consuming unnecessary time and space in core while waiting for either a device to become available or output to be completed. In addition, the device is managed to a better degree because the users cannot tie it up indefinitely, and the demand fluctuations experienced by these devices are equalized.

Second, the DEC-10 has the capability to make maximum utilization of memory. The DEC-10 is a multiprogramming system; i.e., it allows multiple independent-user programs to reside simultaneously in memory and to run concurrently. This technique of sharing memory and processor time enhances the efficient operation of the system by switching the processor from a program that is temporarily stopped because of I/O transmission to a program that is executable. When core and the processor are shared in this manner, each user's program has a memory area distinct from the area of other users. Any attempt to read or change information outside of the area a user can access immediately stops the program and notifies the operating system. Because available memory can contain only a finite

number of programs at any one time, the computing system employs a secondary memory, usually disk or drum, to increase the number of users serviced. User programs exist on the secondary memory and move into memory for execution. Programs in memory exchange places with the programs being transferred from secondary memory for maximum use available main memory. Because the transferring or swapping takes place directly between main memory and the secondary memory, the central processor can be operating on a user program in one part of memory while swapping is taking place in another. This independent, overlapped operation greatly improves system utilization by increasing the number of users that can be simultaneously accommodated.

To further increase the utilization of memory, the operating system allows users to share the same copy of a program or data segment. This prevents the excessive memory usage that results when a program is duplicated for several users. A program that can be shared is called a reentrant program and is divided into two parts or segments. One segment contains the code that is not modified during execution (e.g., compilers and assemblers) and can be used by any number of users. The other segment contains nonentrant code and data. The operating system provides for shared segments to guarantee that they are not accidentally modified.

Third, the DEC-10 has the capability to manage the storage of user program and data files consistent with the multiuser environment. The mass storage devices available are shared among users, and, thus, the operating system must insure independence among the users; one user's actions must not affect the activities of another unless the users desire to work together. To guarantee such independence, the operating system provides a file system for disks, disk packs, and drums. Each user's data are organized into groups of 128-word blocks called files. The user gives a name to each of his files, and the list of these names is kept by the operating system for each user. The operating system is then responsible for protecting each user's file storage from intrusion by unauthorized users. The operating system lets the user specify protection rights, or codes, for his files. These codes designate if others may read the file, and after access, if the files can be modified in any way. Files are assigned protection levels for each of the three classes of users; self; users with a common project number; and all users. Each user class may be assigned a different access privilege, so that there are eight levels in each of the three user classes. This file protection scheme results in a three-digit access code for all files.

1.3.1.3 Constituent Simulations

Within AVSAIL several varied simulations are resident and available to the user. These include: Avionic Evaluation Program (AEP); a general, event-driven hybrid system

simulation program (GASP IV); Distributed Processor/Memory System Network Simulation (DP/M SNS); Software Design and Verification System (SDVS); Avionic System Simulation (AVSIM); Instruction Set Processor (ISP); and the Communication System Evaluation Laboratory (CSEL). These simulations, available in the AVSAIL laboratory, provide a generic simulation capability applicable to all phases of avionic system design and integration. The nature of each one is described briefly in the following paragraphs. More detailed descriptions are given in other sections of this manual.

1.3.1.3.1 Avionics evaluation programs

The interactive Avionics Evaluation Program (AEP) is a collection of avionics performance-assessment models. AEP provides convenient and systematic assessment of avionics in the mission environment. The program is designed to be flexible and easy to use with emphasis on realistic consideration of the operational environment and the generation of useful data. AEP can be utilized for analysis of most air-to-ground and air-to-air missions. Individual programs contained within AEP include air-to-ground and air-to-mission analysis, weapon-delivery error analysis, target acquisition analysis, and a one-on-one dogfight analysis. These programs are implemented in a conversational, interactive mode, thus providing a powerful analytical tool available to users by means of dial-up terminals. They are, therefore, available throughout DOD and to contractor organizations as well.

The air-to-ground mission analysis program evaluates the performance of a flight of up to four aircraft through a specified number of days of operation. The aircraft proceeds along an externally generated nominal trajectory through the mission phases of takeoff, navigation to the search area, search, attack, and return to base. Consideration of ground service requirements is included. Monte Carlo techniques are applied to Mean Time Between Failure (MTBF) data for the defined avionics throughout the mission to determine which subsystem modes are functioning, resorting to backup modes and mission aborts as required.

The weapon delivery analysis routine is a program for determining the distribution of impact errors for a weapon system utilizing unguided, unpowered bombs. The routine is capable of accommodating almost any weapon delivery mechanization under the assumptions of:

1. Flat, nonrotating earth,
2. Linear transformation of component error sources to impact errors, and
3. Normal distribution for all error sources.

The AEP target acquisition model is a modified version of the Multiple Airborne Reconnaissance Sensor Assessment Model (MARSAM II). MARSAM II models the sensor system and the operational environment in detail. It contains models for displays, lenses, filters, and film. It considers the impact of image motion compensation, platform stabilization errors, backscattering, and atmospheric effects on sensor performance. The human observer is modeled in terms of ability to perceive the target as a function of size and contrast, display signal-to-noise ratio, presence of confusing objects, and time in the field of view. Available outputs from MARSAM II include detailed sensor system performance parameters and associated probability measures of detection, recognition, and identification.

The air-to-air AEP analysis program is a Monte Carlo simulation of two opposing aircraft flights (up to four aircraft in a flight) through an entire mission. As the flight progresses, it is influenced by hardware failures, refueling, communications to airborne or ground controllers, enemy aircraft detection capability, identification requirements, and weapon capabilities. When one side detects the other, that flight pursues a course directly at the other flight and fires when the weapon constraints are satisfied. The encounter is considered only until both sides have detected the other. At that time, the relative positions and headings are stored for output so that users can determine which side has the relative advantage.

A separate, deterministic air-to-air program permits analysis of the dogfight encounter. It simulates an engagement between two fighter aircraft. The logic for control of aircraft maneuvers is based on lag pursuit and energy management. Lag pursuit implies that each aircraft attempts to get on the tail of the other. Energy management control implies that the aircraft seeks a velocity and altitude for best turning performance.

1.3.1.3.2 GASP IV simulation language

A simulation language provides the structure and the terminology to facilitate the building of simulations. GASP IV is such a computer language; it helps the user to build computer simulation programs that can be both the model of the system and the analysis vehicle. Thus, this program can be thought of as a model of a system and as a generator of statistical data about the model of the system.

As a programming language, GASP IV gives the computer programmer a set of FORTRAN statements designed to carry out the most important functions in simulation programming. Modeling concepts are translated by GASP IV into FORTRAN routines that can be easily used. GASP IV has five distinct features that make it particularly attractive as a simulation language:

1. GASP IV is FORTRAN based and requires no separate compiling system.
2. GASP IV is modular and can be made to fit on all machines that use a FORTRAN IV compiler.
3. GASP IV is easy to learn since the host programming language is usually known, and only the simulation concepts need be mastered.
4. GASP IV can be used for discrete, continuous, and combined modeling.
5. GASP IV is easily modified and extended to meet the needs of particular applications.

Simulation is divided into two categories: discrete change and continuous change. Note that these terms describe the model, not the real system. In fact, it may be possible to model the same system with either a discrete change (hereafter referred to simply as discrete) or a continuous change (continuous) model. GASP IV is designed to accommodate both categories of models, separately or combined. In most simulations, time is the major independent variable. Other variables included in a simulation are functions of time and are the dependent variables. The adjectives discrete and continuous refer to the behavior of the dependent variables. Discrete simulation occurs when the dependent variables of the model change *discretely at specified points* in simulated time. In continuous simulation the dependent variables of the model may change continuously over simulated time. In combined simulation the dependent variables of a model may change discretely, continuously, or continuously with discrete jumps superimposed. The time variable may be discrete or continuous.

GASP IV is a language that can be used for discrete, continuous, or combined simulation. In GASP IV the most important characteristics of combined simulation, which arise from the interaction between discretely and continuously changing variables, are easily modeled. In general, this interaction takes one of three forms. First, discrete changes may be applied to "continuous" variables. Second, achieving specified conditions for a state variable may cause an event to occur or to be scheduled. Third, the functional description of continuous variables may be changed discretely.

GASP IV specifies that the status of a system be described in terms of a set of entities, their associated attributes, and state variables. The GASP IV simulation philosophy is that a dynamic simulation can be obtained by modeling the events of the system and by advancing time from one event to the next. This philosophy presumes a broader definition of event than has normally been used in discrete-event languages:

> An event occurs at any point in time beyond which the status of a system cannot be projected with certainty.

In GASP IV it is useful to describe events in terms of the mechanisms by which they are scheduled. Those that occur at a specified projected point in time are referred to as time-events. They are commonly thought of in conjunction with "next event" simulation. Those that occur when the system reaches a particular state are called state-events. Unlike time-events, they are not scheduled in the future, but occur when state variables meet prescribed conditions. In GASP IV, state-events can initiate time-events and time-events can initiate state-events.

The behavior of a system model is simulated by computing the values of the attributes at event times. The time-step increment is automatically determined by GASP IV, based on the equation form for the state variables, the time of the next event, and accuracy and output requirements.

The key to event simulation is the ability to organize events so that they are executed within the computer in an order corresponding to that which would occur in the real system. This preserves the time relationship between simulated and real events. Ordinary programming languages are unsuited to this task because they operate in a strictly sequential manner; there is no way to tell a FORTRAN program to "do something later'. without building special subprograms. GASP IV provides these subprograms.

Every GASP IV simulation model consists of: (1) a set of event programs or state variable equations, or both, that describe a system's dynamic behavior, (2) lists and matrices that store information, (3) an executive routine that directs the flow of information and control within the model, and (4) support routines. These form an operating computer program whose performance reflects that of a simulated system. A GASP IV program is made up of subprograms linked together by an executive routine that organizes and controls the performance of the subprograms.

GASP IV is organized to provide eight specific functional capabilities:

1. Event control,
2. State-variable updating using integration if necessary,
3. Information storage and retrieval,
4. System state initialization,
5. System performance data collection,
6. Program monitoring and event reporting,
7. Statistical computations and report generation, and
8. Random deviate generation.

The functions provide the user with a very general tool with which to build simulations. For example, GASP IV language provides the basis for the Basic Simulator (SIMNUC) described below, which in turn has been used to build the Distributed Processor/Memory simulation available at AVSAIL.

1.3.1.3.3 Basic simulator (SIMNUC)

The Basic Simulator (SIMNUC) is an integrated package of subprograms designed to facilitate modeling and simulation of discrete stochastic systems in a manner similar to the GASP IV simulation programs.

The following features characterize this package:

1. Model independence.
2. FORTRAN orientation; the user's portion of a simulator can be programmed in FORTRAN or, if desired, in assembly language.
3. Capability to produce event-oriented simulation models.
4. Availability of list processing and dynamic memory management facilities.
5. Capability to collect and display standard queue and sample statistics.
6. A full complement of random number generators.

The basic approach, which sometimes is referred to as a simulation-world-view, used to model discrete systems for digital simulation with the Basic Simulator is the event-oriented approach, which emphasizes decomposition of the simulation process into individual event procedures, each of which describes all changes in the system caused by the occurrence of the related event, just as was done in GASP IV.

The Basic Simulator consists of the following functional software components:

1. Dynamic memory management,
2. List processing,
3. Simulation run control,
4. Random number generators,
5. Sample statistics processing, and
6. Error diagnosis and reporting.

1.3.1.3.4 Distributed processor/memory system network simulation

The advent of the minicomputer and the microprocessor has made available to the avionics system designer highly compact and versatile computational capabilities that can be both physically and functionally distributed among avionics equipments. Used in conjunction with multiplexed data buses, these processors make up distributed processing systems presenting new challenges to the system designer.

The Distributed Processor/Memory (DP/M) System Network Simulator (SNS) provides the necessary tool to explore some of the tradeoffs available to designers of these distributed systems. The SNS is a discrete, event-oriented, high-level traffic simulator written in ANSI standard FORTRAN. The SNS is built around a nucleus of model-independent utility routines (SIMNUC) which are not simulators in themselves, but are used to create a simulator in conjunction with the avionic software task specifications and topological organization specifications of a given avionics system.

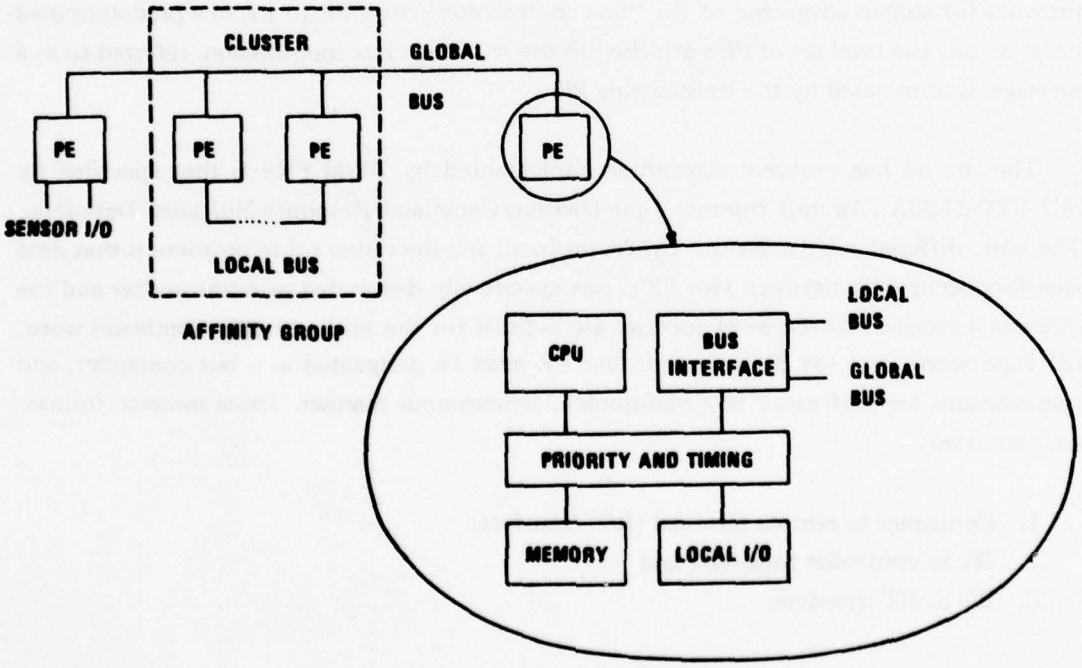


Figure 1.3.1.3-1. DP/M system architecture.

The DP/M system concept is essentially the use of varying numbers of simple, homogeneous processor/memory elements (PE's) applicable to a wide range of avionic system processing problems. Architecturally, these PE's can be used as stand-alone uniprocessors, or they can be configured in a distributed network as shown in Figure 1.3.1.3-1. Serial-time-division-multiplex (TDM) buses interconnect the network. Two levels of busing are provided: a Global bus can interconnect each PE in a system network, and a Local bus can interconnect multiple PE's clustered together to perform a given function. This cluster of PE's is referred to as an Affinity Group (AG). Input/output (I/O) for a given PE to an external device is via its local I/O interface unit.

The SNS, being constructed of SIMNUC model-independent routines, has the same general characteristics as SIMNUC and GASP IV. That is, SNS is a discrete, event-oriented simulation system. Unlike a continuous system where transitions from one state to the next are a continuous function of time, transitions from one state to another in a discrete system occur at discrete points in time. Distinguishable state transitions are called events. Event-oriented simulation systems emphasize a detailed description of the steps that occur when an individual event takes place.

During the period the DP/M SNS has been in use at AFAL, two bus protocol algorithms have been implemented. The first is a modified "round-robin" slotting technique, which provides for simple advancing of the "bus control slot" from PE to PE in a predetermined order among the total set of PE's attached to the bus. Each bus transmission, referred to as a message, is terminated by the transmitting PE.

The second bus protocol algorithms implemented by DP/M SNS is that specified by MIL-STD-1553A (Aircraft Internal Time Division Command/Response Multiplex Data Bus). The basic difference between the 1553A protocol and the round-robin protocol is that data transfers occur only between two PE's, one specifically designated as a transmitter and the other as a receiver. Three word formats are defined for the protocol: (1) Command word, (2) Data word, and (3) Status word. One PE must be designated as a bus controller, and transmissions are performed in a half-duplex, synchronous manner. Three message formats are permitted:

1. Controller to remote terminal (RT) transfers,
2. RT to controller transfers, and
3. RT to RT transfers.

The hardware architecture represents one half of the distributed avionics system design problem. The other half is the software, obviously. The DP/M SNS assumes that the system

designer will partition the executive and applications software into suitable tasks for some given hardware architecture. The SNS then provides the designer with the capability to analyze and evaluate:

1. Processing element characteristics/capabilities,
2. Number of resources in the system,
3. Local and global bus configuration,
4. Inter-PE communication technique,
5. PE Bus protocol communication technique, and
6. Executive control technique.

The SNS is predicated on the representation of software modules by a directed graph consisting of a set of nodes and a set of directed edges between these nodes. A node is used to represent a set of computations which, once initiated, can run to completion without waiting for completion of another set of computations also represented by a node. An edge from node i to node j means that, upon completion of the computations represented by node i, the computations by node j can be initiated.

The representation of software execution sequences via a directed graph has a particular advantage within the DP/M concept. The subfunction-directed graph reveals potential process construction options in allocating tasks and program to PE's. If any one set of tasks must be partitioned among several PE's, the options available in allocating this software to PE's are clearly defined within the graph. Data sets that are passed from one task to another represent Local bus messages if their respective tasks are not collocated in the same PE. Likewise, collocated tasks need not generate bus traffic with their data interchanges.

The use of distributed PE's in the DP/M system concept dictates the need for a method of scheduling activities, transferring bus messages between PE's and general system control. These operations are referred to as Executive functions and are provided by the SNS. The subfunction-directed graph contains the necessary information from which the Executive can determine task-scheduling conditions (based upon required predecessor events) and intertask communication. The DP/M Executive structure provides two levels of control: the Global Executive (GEX) and the Local Executive (LEX). Functionally, the GEX assumes the role of system monitor and scheduler. It enforces subfunction interrelationships and is responsible for system performance by coordinating those software programs required to effect mission avionic functions for the pilot and aircraft. The LEX is a PE-oriented function responsible for sequencing and controlling tasks assigned to a PE. The LEX is concerned with scheduling those tasks assigned to its PE, based upon successful satisfaction of all the tasks' given predecessor conditions.

The Global Executive schedules time-dependent subfunctions in the DP/M system. A time-ordered linked list provides the GEX with the relative times to schedule every time-dependent subfunction in the system. A "go" message is generated by the GEX to a subfunction only if other predecessor conditions for the subfunction have been satisfied when it is time to run the subfunction. The GEX data base (or tables) contains all information pertaining to the initialization and control of all time-dependent subfunctions in the DP/M system.

A family of data collection and report generation programs is provided with the DP/M System Network Simulator. These programs provide the capability to selectively collect data on and generate reports for the various system parameters under investigation for a particular DP/M system configuration and/or avionic mission segment. The collection and dispensation of data, as well as generation of reports, are controlled by user specified parameters. In general, the user has four options: (1) no data is collected and no reports generated; (2) data is collected and saved, but no report generated; (3) data is collected and report generated, but data not saved; or (4) data is collected and saved, and reports generated. Data saved on tape may be processed at a later time. In fact, this saved data may be used at a later time to compare results of two or more different simulation experiments.

Data collection and report generation occur at three distinct levels: (1) event level, (2) sample period level, (3) postsimulation level. At each of these levels, reports concerning bus performance, processor loading, executive performance, and number of avionic tasks processed may be selectively generated.

1.3.1.3.5 *Software design and verification system (SDVS)*

Software tools to aid in the development, testing, verification, and maintenance of avionic mission software are provided by SDVS. Simulations available under SDVS are non-real-time. The SDVS was created as an integral part of the DAIS program, and the user will encounter some constraints imposed by the DAIS concept. Since the use of common hardware and software for acquisition of sensor data, processing of information, and provision of display information is key to the DAIS concept, the software design and verification functions, in the context of any particular processor architecture, are vital to success. These same considerations for software integrity are common in some degree to all avionic software developments. Thus, SDVS can play an important role in software development.

The basic functions provided by SDVS are depicted by Figure 1.3.1.3-2. Configuration Management refers to provisions for control of all files associated with the development,

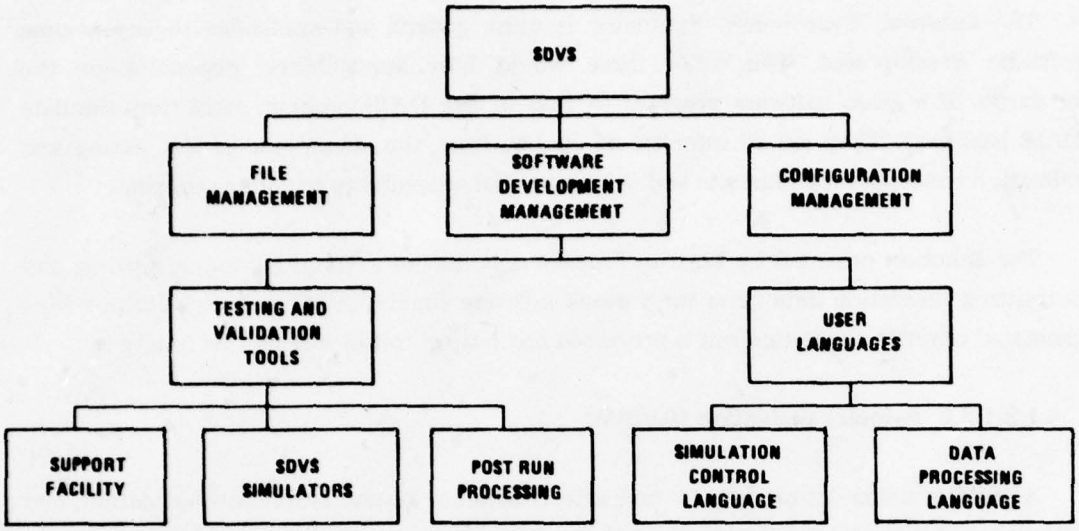


Figure 1.3.1.3-2. SDVS functional organization.

test, and verification of software. An extensive cataloging and security system is provided for the various types of software maintained by SDVS. A set of interactive conversational commands is provided by the File Management function; these commands enable the user to perform various file manipulation activities necessary during software development. In interpreting these commands, the Configuration Management catalogs are interrogated to determine if the user has file-access authority and, if so, the type of access authority.

Software Development Management functions are divided into two groups. In order to test software efficiently and comprehensively, an easy-to-use man/machine interface is necessary to facilitate the specification of the simulated environment, data to be recorded, and the required processing of simulation data. Two special purpose languages, the Simulation Control Language and the Data Processing Language are provided by SDVS for these purposes. Testing and validation of software are facilitated by several tools provided by SDVS, the first being the support facility function, which is DAIS-specific and not directly applicable to other software development programs. A second testing and validation software function is provided by four simulators. These simulators model processor and bus performance in the execution of mission software:

1. Statement Level Simulator,
2. Interpretive Computer Simulator,
3. Data Bus Simulator, and
4. External Environment Simulator.

The External Environment Simulator is quite general and applicable to any avionic software development. The other three would have applicability, depending on the similarity of a given software program to that of the DAIS program since they simulate DAIS hardware. They are of interest, of course, from the standpoint of the testing and validation concepts they illustrate and their potential adaptability to other programs.

The function provided by Postrun Processing is that of sorting, editing, analyzing, and outputting simulation data from the various software simulations. The Rough Output Tape generated during a simulation run is processed and output to the user by this function.

1.3.1.3.6 Avionics simulation (AVSIM)

AVSIM is a simulation facility that affects avionics system evaluation, validation, and integration by dynamic digital simulation of the airframe, flight controls, and avionic equipments of a generic high-performance tactical fighter. Currently, the objectives of this facility are:

1. To test and validate operational flight programs under realistic flight conditions,
2. To affect digital avionics system integration,
3. To identify hardware/software problems in prototype avionics systems, and
4. To recreate flight problem areas through dynamic simulation.

AVSIM is capable of simulating the navigation, penetration, and weapon-delivery phases of an attack fighter mission, either individually or compositely. The AVSIM user configures his aircraft, sensor complement, environmental characteristics, and target characteristics by linking individual simulation models into an overall simulator structure. The AVSIM simulator currently has real-time, non-real-time, man-in-the-loop, and self-contained modes of operation. The user has the option to use resident (F-16) software developed by General Dynamics, Ft. Worth; to use resident (A7) software obtained from the Navy; or to develop his own by utilization of resident creation routines.

In the example of the resident software, the airframe is configured by selecting either an F-16 or an A7 aircraft model; an appropriate flight control system dependent on desired complexity; and self-contained (synthetic mission generator/simulated pilot), prerecorded, or real-time cockpit inputs. The sensor complement presently available includes the radar altimeter, the attack radar, and (may be extended to include) electro-optical sensors. Flight environment is incorporated by using models that provide simulated air data inputs, accelerometer and gyro outputs, representative weather effects, atmospheric perturbations,

inertial outputs, and magnetic heading. Auxiliary software integral to the total simulation includes an inertial reference, geometry effects, and the ability to introduce noise at various points.

AVSIM is hosted by the DEC-10 facility at AFAL and is linked to peripherals such as the cockpit and display generator by means of a DMA channel to satellite PDP-11's. AVSIM software consists of control modules and application models. The control software provides file manipulation, sets up the simulation configuration, provides initialization, implements man/machine interface, and controls overall the execution sequence. The application models provide aforementioned sensor data, external physical conditions, etc. Also contained within the software are data acquisition and analysis modules, which accumulate and edit data for validation analysis.

AVSIM programs are coded largely in ANSI-FORTRAN in an attempt to make the software flexible, modular, and transferable. DEC system FORTRAN-10 features as well as DEC-10 assembly language are also used to a lesser degree.

1.3.1.3.7 Processor architecture (ISP)

The ISP processor simulation facility enables a user to describe computer processing units at the register transfer level and, from these descriptions, to quickly set up interactive simulations of these processors. A language, CSL/ISP (Computer Simulation language) was developed as a dialect of Instruction Set Processor language (ISPL) from Carnegie-Mellon University. A compiler produces DEC-10 code from the CSL/ISP source, and the user then runs this code from a control program, which he constructs by modifying a model general purpose simulation control program. A methodology is included for creating simulations from manufacturers' instruction set descriptions.

The processor simulation program can, from a description of the register transfers of a computer, produce a simulation of that computer. This program may be broken into three general areas. These include: a formal language compiler with which to describe register transfer level processes, a code generator to produce DEC-10 code from the output of a compiler for that language, and a general purpose control program with which to drive the simulators produced from the compiler.

The compiler uses a set of register transfer operators, called XT-op's, to produce a set of pseudo-instructions, which can be interpreted by the DEC-10 assembler's (MACRO) macro facility. The macros, which were written to produce DEC-10 code from these

pseudo-instructions, produce optimized code and comprise a universal library, which is searched during assembly.

Simulations are controlled by a program with which the user can interactively set registers and memory locations, load memory, set breakpoints, etc. The control program causes instructions in simulated memory to be executed by repetitively calling on the simulator to execute a single instruction.

The behavior of a processor is determined by the nature of its individual operations and the sequence in which those operations occur. This sequence is generally governed by a stored program, which resides in the memory of the computer and the set of interpretation rules which the processor applies to the program.

Although the above format is commonly used to describe digital computers, ISP does not limit the user to a particular type of description. Thus, ISP can be used to describe register transfer systems in general; digital computers are a subset of such systems which interpret an instruction set. Other devices, such as buses and device controllers can also be described in ISP.

1.3.1.3.8 Communication system evaluation laboratory (CSEL)

CSEL is a combined hardware/software facility designed to analyze, synthesize, and model advanced communication systems. The laboratory centers about a computer-based simulation facility, which is capable of creating a variety of hostile RF signal environments at UHF and L-, X-, and K-band. To this facility may be interfaced, for testing and evaluation, either laboratory-model communication hardware, actual communication hardware, or a combination of elements of both. To aid in the construction of laboratory communications systems, CSEL provides a high-speed, programmable signal processor and a spectrum of communication equipment, including modems, terminals, and antenna systems.

It is important to note that the simulation facility just mentioned, called the Signal and Interference Generator, produces simulated signal environments in the appropriate RF band. Thus, it qualifies as a hardware simulator, control over which is exercised dynamically by a digital computer operating in real time. Initial configuring of the simulator is also performed through the digital controller by means of a series of user commands, which the system software interprets and translates into control signals to the communication hardware.

Interfacing communication terminals to the Signal and Interference Generator RF hardware then provides a realistic test bed for the terminals, in which one can not only

troubleshoot the equipment but also test its performance with respect to such environmental effects as jamming and fading. The user can specify these effects with relative ease and can vary them readily from one test run to the next, thereby obtaining a complete characterization of the performance capabilities of his equipment.

A typical application of CSEL is illustrated by the tests performed to determine the suitability of the LES-8/9 communication satellites for use with the Airborne Command Post. In these tests, CSEL was used not only to troubleshoot Ka-band communication hardware, but also to ascertain the vulnerability of the communication system to various types of jamming.

To accomplish these goals, CSEL was equipped with appropriate K-band and UHF communication modems, terminals, and antennas. This configuration allowed use of the communication satellite in a laboratory system equipped with a qualification model of the AN/ASC Ka-band airborne communication terminal, together with antenna systems, to establish communication links with LES-8/9. This equipment, when combined with the Signal and Interference Generator, allowed the realization of actual satellite communication channels into which were introduced controlled-interference efforts in the form of jamming, fading, and doppler. To reinforce this capability, the high-speed signal processor, called the Programmable Data Terminal, was programmed to simulate satellite processing when LES-8/9 were unavailable to use.

Current emphasis in CSEL is shifting toward a study of the performance of communication systems linking remotely piloted vehicles with air- and ground-based command posts. To this end, the facility is being upgraded to include the elements of a video processing and display capability. Future studies envisioned for CSEL involve satellite-based navigation systems and the performance they obtain in a hostile environment.

1.3.2. Electronic Technology Division

To be included in a future version of this manual.

1.3.3 Electronic Warfare Division

To be included in a future version of this manual.

1.3.4 Reconnaissance and Weapon Delivery Division

To be included in a future version of this manual.

SECTION II

AVIONIC SYSTEM ANALYSIS AND INTEGRATION LABORATORY (AVSAIL)

2.1 AVSAIL HOST FACILITY

The AVSAIL host facility has been configured particularly for implementation of all the simulation classes previously described. In order to convey the flexibility and power of the AVSAIL laboratory, the host facility is described in the following sections. The scope of the descriptions is limited to those aspects of the facility which have significance to the conduct of simulations, rather than to an exhaustive exploration of capability. Discussions are provided of the basic computer and peripheral hardware configuration and performance, the operating and utility software capabilities, and the communication interfaces to onsite simulators and offsite users.

The concept of AVSAIL encompasses all of the analysis and integration functions necessary to create innovative avionics systems. These functions are best illustrated by the discussion of the simulation class structure in Section 1.2 of this manual. One of the more important aspects of the AVSAIL concept is the need for a simple interactive interface for the user of the laboratory host facility. The complex nature of the problems which may be investigated in this laboratory imply many users over extended periods of time. The systems being designed will be subjected to evolutionary modifications, necessitating comparisons of performance of components and systems at various stages of the design. In order for the AVSAIL facility to provide the degree of interaction and flexibility required to simulate a wide range of systems and operational scenarios, the facility architecture of Figure 2.1-1 was proposed.

This architecture demonstrates the explicit requirement that the user be allowed to address the simulation facility without regard to facility management and control constraints. This is achieved by use of appropriate interface structure, which translates user requirements into facility inputs interactively. The focal point of the simulation facility architecture is the simulation control program, which addresses the various simulation levels and handles the control and response files according to user requirements.

A set of simulation models is available to the user in the model data base. Models currently available are described in this manual and others may be added as required. The user may modify model parameter values for his particular simulation needs, modify models, or create his own models and add them to the data base. Such modifications or additions to the data base would be under the supervision of the facility manager, however.

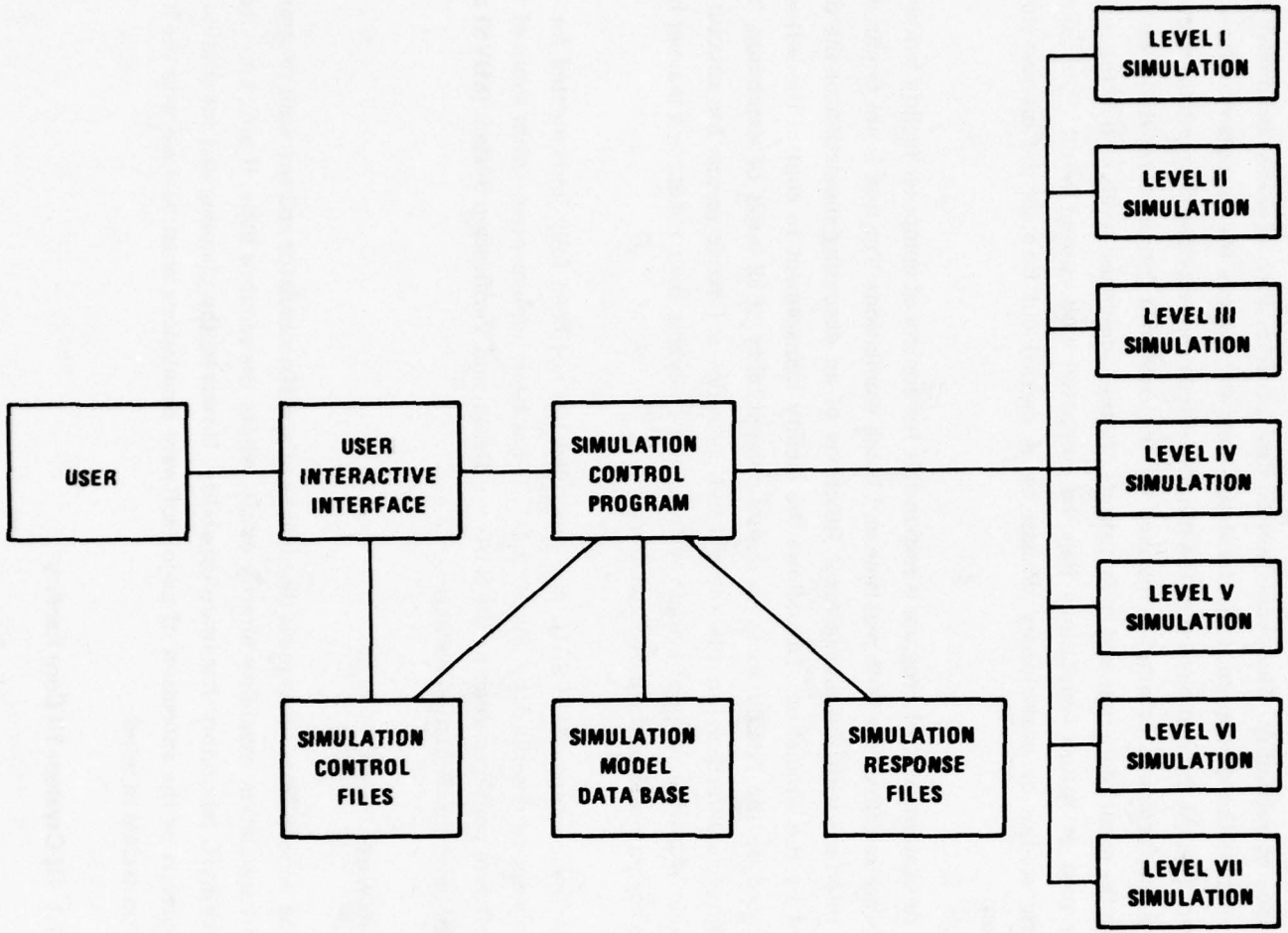


Figure 2.1-1. AVSAIL architecture.

In order to simulate a system or component, the user would create a simulation control file that would specify such items as flight scenarios, system configuration, model selection, model parameters, simulation time frames, data recording, data analysis, and component performance or reliability. The simulation control program then accesses the model data base, configures the simulation, runs the simulation, and records the outputs as defined by the control file. At the completion of the run, the simulation response file contains a record of system performance parameters defined by the user in a format compatible with all models in the model data base and with outputs of other simulation levels so that the results of other past or future simulations may be compared with current results. This allows parametric studies or evolutionary designs to be carried out through the mechanism of simulation.

The simulation control program is responsible for control of computer facility hardware and software resources for both real-time and batch simulations. The user is not required to directly interface with the management functions of an executing simulation or the data generated by the simulation. This allows the facility management to control the software implemented on the facility so as to ensure compatibility of all levels of simulation. The user interactive interface, on the other hand, provides a flexible means for structuring simulations of a wide range of avionic systems and analyzing their results in a manner best suited to the purposes of the individual users.

While the concept of AVSAIL just described has not been fully implemented for the overall system as described by Figure 2.1-1, it has been implemented within some of the major software packages, such as the Software Design and Verification System (SDVS) and the AVSIM avionic simulation package.

2.1.1 Hardware

As can be seen from subsequent discussions of specific simulator and software programs, significant simulation capability already exists, whose use requires little, if any, knowledge of the AVSAIL laboratory hardware capability. However, the planning and integration of new simulators or the execution of major software simulations must be done with the host facility constraints in mind.

2.1.1.1 DECsystem-10 Core Facility

2.1.1.1.1 System description

The AVSAIL laboratory is structured around a Digital Equipment Corporation DEC-10

mainframe computer. A wide range of peripheral input/output devices, as described subsequently, are provided to handle batch, timeshare, and real-time program development and execution requirements. A unique feature of the AVSAIL configuration is the eight-channel Direct Memory Access (DMA) capability provided for access by a complement of eight PDP-11 series minicomputers. The PDP-11's serve to interface specific simulations (e.g., the F-16 Fire Control Computer simulator) to the DEC-10. An overall hardware system diagram is provided in Figure 1.3.1.1-1, and the major elements are briefly described below.

2.1.1.1.1.1 DEC-10 Central Processor and Main Memory. The central processing unit (CPU) for the AVSAIL DEC-10 consists of dual KI-10 processors, with the configuration being designated as a DEC-1077. Each CPU module is an independent processor, and programs can operate in parallel, one program per processor, thereby increasing the average throughput speed. Other performance characteristics of the DEC-1077 are given in Table 2.1.1.1-1.

As shown in Figure 1.3.1.1-1, the DEC-10 memory is made up of 4 DEC 64K modules, and an additional 512K words of Ampex ARM-10LX memory, for a total of 768K words. Word size is 36 bits plus parity. Each of the memory modules, as well as the CPUs and data channels, are interfaced via the memory bus. The structure of the memory bus gives the central processor and high-speed data channels simultaneous access to separate memory modules and allows each to operate at its own speed. The memory bus system allows each data channel to transmit full 36-bit words in parallel at a speed of 1 million words per second. In total, the memory structure operates at rates of up to 10 million characters per second when I/O devices and processors are simultaneously transferring data.

TABLE 2.1.1.1-1. DEC-1077 PERFORMANCE

Memory size (min - max)	128-4096K	Instruction times (μs)	
No. of instructions	378	Fixed point add	1.5
Instruction look-ahead	Yes	Fixed point multiply	4.1
Virtual memory	Yes	Jump	1.1
Memory interleaving	2 or 4 way	Single precision floating point added	3.6
Index registers	4X15/CPU	Double precision floating point added	7.6
Accumulators	4X15/CPU	I/O bus band width, words/s	370K
		Memory bus bandwidth words/s	4000K

2.1.1.1.1.2 Bulk Storage. Bulk storage is available both on multisurface cartridge disks and magnetic tape. All disk and tape units are interfaced to both the I/O bus and to the memory bus.

The system currently provides four RPO3 disk drives and four RPO4 disk drives. The RPO3's and RPO4's are separately interfaced as shown by Figure 3.1.1.1-1. Some performance characteristics of these disk drives are given in Table 2.1.1.1-2. Storage capacity of the RPO4's is twice that of the RPO3's, and transfer rate is nearly triple that of the RPO3's.

Magnetic tape drives available also provide a choice of performance. Four TU10A-E 9-track drives operate at 45 ips and four TU 40 drives operate at 150 ips, providing a range of transfer rates from 9K to 120K characters per second. Additional performance data is given by Table 2.1.1.1-3.

2.1.1.1.1.3 User Interface. The DEC-10 facility provides for onsite and offsite access for both development of software and for its execution. Currently, onsite facilities include 25 CRT alphanumeric terminals, 2 local CRT graphic display terminals and 2 minicomputer based data acquisition systems. In addition to these physical terminals, the DEC-10 monitor

TABLE 2.1.1.1-2. DISK DRIVE SYSTEM CHARACTERISTICS

Characteristic	RP03 Disk	RP04 Disk
Disk drive capacity	10.24 million words	20.48 million words
Transfer rate	15 μs/word	5.6 μs/word
Access time:		
Track-to-track	7.5 ms	7 ms
Average	29 ms	28 ms
Maximum	55 ms	50 ms
Organization:		
	128 words/sector	128 words/sector
	10 sectors/track	20 sectors/track
	20 tracks/cylinder	19 tracks/cylinder
	400 cylinders/pack	411 cylinders/pack
Number of heads	20	19
Number of recording surfaces	20	19
Number of disks	11	12
Number of drives/controller	8	8
Number of drives/system	32	32
Maximum storage/system	1.96 billion characters	3.92 billion characters

TABLE 2.1.1.1-3. MAGNETIC TAPE DRIVE CHARACTERISTICS

Characteristic	TU10A-E	TU40
Tape speed	45 ips	150 ips
Transfer rate at:		
220 bpi	9K char/s	30K char/s
556 bpi	25K char/s	83.4K char/s
800 bpi	36K char/s	120K char/s
Recording technique	NRZI	NRZI
Nomial interrecord Gap:		
9 track	0.6 in.	0.6 in.
Rewind time (2400 ft)	195 s	66 s

software support up to 48 virtual terminals or "pseudo-teletypes" which allow jobs to control other jobs. Normally a job is associated with a physical terminal, but the pseudo-teletype function allows jobs to be initiated and controlled by other jobs. A controlling job sends the same kinds of commands to its subjobs as would be sent by a user at a physical terminal, and the monitor does not distinguish jobs controlled by a physical terminal from those controlled by a pseudo-teletype. The physical user interface is thus expanded by software to increase system throughput potential. The physical terminals just described are interfaced to the DEC-10 through DC76-DA Data Communications System (Figure 1.3.1.1-1). Asynchronous lines operating at 300 baud connect to the terminals, although interface capability is for 9,600 baud maximum line, and a maximum aggregate transfer rate of 30,000 bps.

Off-site access to the DEC-10 is through the DC75 Data Communications System (Figure 1.3.1.1-1) which provides a maximum of eight synchronous, 9,600 baud lines. Maximum aggregate data rate is 30,000 bps. Currently, 4,800 baud full-duplex leased lines connect the AVSAIL laboratory through this interface with the Armament Development Center, Eglin AFB, Florida, and the Naval Weapons Center, China Lake, California.

2.1.1.1.1.4 Hard Copy Devices. Two line printers provide high speed output from the DEC-10. The 1,250 line per minute LP10FA is a 132 column format printer with a 64 character EDP font. The LP10HC provides the same performance, but has a 96 character scientific font. Graphical hard copy output is available from a Calcomp Model 563 Plotter. This high speed, drum-type pen and ink plotter uses 31 in. paper and operates at up to 300 steps per second. Card input to the system is available through a CR10E 1,200 card per minute reader.

2.1.1.1.2 DECsystem-10 processor features

In order to convey some of the power of the DEC-10, some of the more important features are described in the following paragraphs. These features will be of more value to the user initiating a major new simulation than to those using or modifying slightly existing simulations.

2.1.1.1.2.1 Instruction Set. The KI10 has 378 instructions, an extremely large repertoire which provides the flexibility required for specialized computing problems. Since the set provides so many instructions to choose from, few instructions are required to perform a given function. Assembly language programs are therefore shorter than with other computers, and the instruction set simplifies the Monitor, language processors, and utility programs. For example, compiled programs on a DEC-10 are often 30 to 50 percent shorter, require less memory and execute faster than those of comparable computers.

In addition to these instructions, the DEC-10 provides 64 programmable operators, 33 of which "trap" to the Monitor (Monitor calls) and 31 of which trap to the user's core area. The remaining instructions are unimplemented and reserved for future expansion. An attempt to execute one of these unimplemented instructions results in a trap to the Monitor.

The instruction set, despite its size, is easy to learn. It is logically grouped into families of instructions and the mnemonic code is constructed modularly. All instructions are capable of directly addressing a full 256K (36-bit) words of memory without resorting to base registers, displacement addressing, or indirect addressing. Instructions may, however, use indirect addressing with indexing to any level. Most instruction classes, including floating-point, allow immediate mode addressing, where the result of the effective address calculation is used directly as an operand in order to save storage and speed execution.

The half-word data transmission instructions move a half-word and may modify the contents of the other half of the destination location.

The full-word data transmission instructions move one or more full words of data from one place to another. The instructions may perform minor arithmetic operations such as forming the negative or the magnitude of the word being processed. The five byte manipulation instructions pack or unpack bytes of any length, anywhere within a word. The logic instructions provide the capabilities of shifting and rotating, as well as performing the complete set of 16 Boolean functions on two variables.

The KI10 processor implements instructions to perform scaling, negating, addition, subtraction, multiplication, and division upon numbers in single-precision and double-precision floating-point format. In the single-precision floating-point format, 1 bit is reserved for the sign, 8 bits are used for the exponent, and 62 bits are used for the fraction.

Special KI10 instructions provide the capability of converting fixed-point formats to or from floating-point formats. Two sets of instructions are provided to perform this function: one set optimized for FORTRAN and a second set optimized for ALGOL.

The arithmetic testing instructions may jump or skip, depending on the result of an arithmetic test and may first perform an arithmetic operation on the test word. Instructions are also provided to modify and/or test using a mask and/or skip on selected bits in an accumulator. Program control instructions include several types of jump instructions and the subroutine control PUSHJ and POPJ instructions.

Input/output instructions govern all direct transfers of data to and from the peripheral equipment and also perform many operations within the processor. Block transfer instructions handle bulk data transfers to/from I/O devices.

The KI10 has hardware for processing both single-precision and double-precision floating-point numbers. There are eight double-precision instructions and three fixed/floating conversion instructions. A double-precision word consists of the sign, an 8-bit exponent and a 62-bit fraction. This gives a precision in the fraction of 1 part in 4.6×10^{18} and an exponent of 2 to a power of from -128 to +127.

2.1.1.1.2.2 Processor Modes. Instructions on the DEC-10 are executed in one of two modes depending upon whether a mode bit has been set. Programs operate in either User Mode or Executive Mode. In Executive Mode operations, all implemented instructions are legal, addresses are not relocated, and all core locations are accessible. The monitor operates in Executive Mode and is able to control all system resources and the state of the processor. In User Mode operations, addresses are relocated, certain instructions are illegal, causing monitor traps when executed, and address references are confined within two program segments.

The KI10 further divides Executive and User Mode operation into two submodes each. User Mode is subdivided into public and concealed submodes and Executive Mode into supervisor and kernel submodes. For each 512-word page in the system, information is stored in a table maintained by the operating system which specifies whether a page can be accessed or altered, and if it is defined to be public or concealed. The Executive and User

Modes subdivide on the KI10 according to whether the active program is running in a public or concealed area. Within User Mode are the public and concealed submodes; within Executive Mode, the supervisor and kernel submodes. These mode features are summarized in Table 2.1.1.1-4.

2.1.1.1.2.3 Processor Memory Management. The KI10 provides memory address mapping from the program's memory address space (referred to as the effective address) to the physical memory address space by substitution of the most significant bits of the memory address. This mapping provides access to the entire physical memory space which is 16 times larger than the maximum user address space. The user's effective address space is 256K words addressed with 18-bit addresses; the physical address space is 4,194,304 decimal.

The memory mapping process utilizes the most significant nine bits of the effective address as an index into the appropriate page map (User or Executive) in memory. The data located by the index provides 13 bits which are appended to the least significant 9 bits of

TABLE 2.1.1.1-4. PROCESSOR MODES

User Mode	
Public Submode	**Concealed Submode**
• User programs • 256K word address • All instructions permitted unless *they compromise integrity of* system or other users • Can transfer to concealed submode only at entry points	• Proprietary programs • Can READ, WRITE, EXECUTE, or TRANSFER to any location labeled Public
Executive Mode (Monitor)	
Supervisor Submode	**Kernel Submode**
• Performs general management of system • Performs those functions that affect *one user at a time* • Executes in virtual address space labeled Public	• Performs I/O for system • Performs those functions that affect all users

the effective address in order to form the 22 bit physical address. Also provided are three bits which indicate what type of memory requests are allowed to the page in question (one, read-only, proprietary, etc). If this scheme were implemented exactly as outlined above, every user memory reference would require two actual memory references: one to obtain the memory mapping data and another to obtain the user's mapped memory reference. In order to reduce the number of actual memory references to nearly the same number as required by the program, an associative memory mapping unit is used in the KI10 as illustrated by Figure 2 1 1.1-2.

The Monitor assigns the core area for each user by loading the various page tables, setting up the trap locations in the user page map, and responding appropriately when a trap occurs. The Monitor provides memory protection for itself and each user by filling the page tables only with those entries which are allowed to be accessed. A zero access bit in the page table will cause reference to the associated page to initiate a page failure trap to the Monitor. The TOPS-10 Operating System utilizes the KI10 and page maps to create one or two segment programs. The major benefits of the paging capability are a smaller unit of core allocation (512 words instead of 1,024), the freedom to scatter the pages of a segment randomly through physical core (avoiding core fragmentation and the overhead of repacking core), and the opportunity to execute a program when all of its pages are not in physical core (i.e., a virtual memory capability).

2.1.1.1.2.4 Real-Time Clock. The DK10 Real-Time Clock provides high-resolution timekeeping for time accounting, time-base maintenance, periodic high-frequency inter-

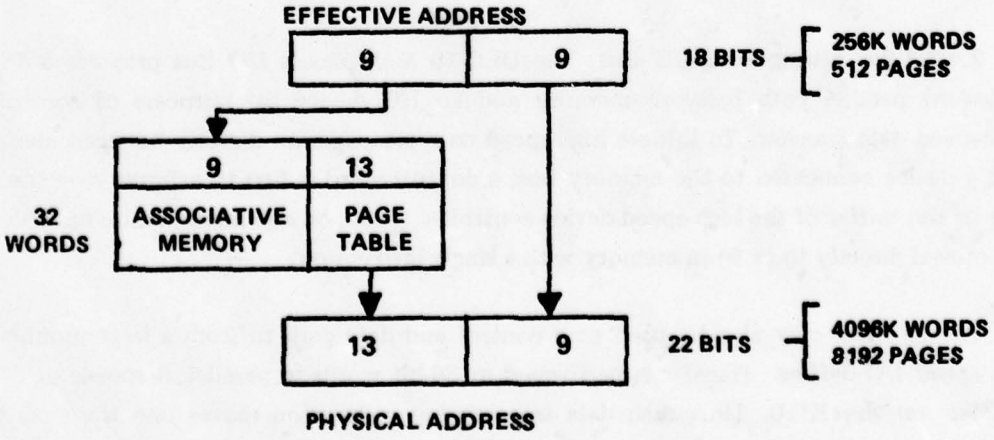


Figure 2.1.1.1-2. Address computation scheme for KI10 processor.

rupts, and interval timing. The clock provides 110 μs resolution and a choice of up to 2^{18} possible timing intervals, so that interrupts can be programmed at intervals from 10 μs up to 2.6 sec.

In addition to an interval register, the DK10 has a frequency counter which counts the pulses of an internal ± 0.01 clock kHz, or an external clock having a maximum frequency of 400 kHz. The clock also includes a comparator network which provides a running comparison between the frequency counter and the interval register. When the frequency counter reading equals the total on the interval register, a program interrupt is generated and the frequency counter is automatically reset so that it can time the next interval.

The clock, which is assignable to any interrupt channel, can be used to pace real time, monitor, or other functions performed in either Executive or User Modes. Clock updating is interlocked with the DATAI instructions so that it can be read correctly at any time by the KI10 without losing a clock pulse.

2.1.1.1.2.5 Fast Register Blocks. General-purpose registers are another DEC-10 feature that help improve program execution. These sets of fast integrated circuit registers can be used as accumulators, index registers, and as the first locations in memory. Since the registers can be addressed as memory locations, they do not require special handling instructions. Four sets of 16 fast registers are included in the KI10. Program switching time for the KI10 between register stacks is 2.5 μs. On the KI10, different register blocks can be used for the operating system and individual users. This eliminates the need for storing register contents when switching from User Mode to Executive Mode. Also, a critical real-time program is able to maintain its own register block for handling data and interrupt sequences at maximum speed.

2.1.1.1.2.6 Multiplexed I/O Bus. The DEC-10 Multiplexed I/O Bus provides a 36-bit full-word parallel path between memory and an I/O device for purposes of control or low-speed data transfer. To initiate high-speed data transmission directly between memory and a device connected to the memory bus, a control word is first transferred over the I/O bus to the buffer of the high-speed device controller. Then, on command, entire data blocks are moved directly to or from memory with a single instruction.

The I/O bus may also be used as a control and data path to/from a large number of low-speed I/O devices. Transfer is performed in 36-bit words in parallel at speeds of 370K words/s on the KI10. Thus each data transmission instruction moves one word of data between memory and the buffer of the device controller. When block input or output instructions are used, entire blocks of data are moved to or from the device with a single instruction.

2.1.1.2 Direct Memory Access of Simulators

The AVSAIL laboratory is designed to facilitate testing of hardware/software systems in a simulated real-world environment. The systems under test may include hardware and software only, or, alternatively, human (pilot) interaction with a simulated external environment may be included as a system component. An example might be the testing of a fire control computer and its software with a human pilot flying in a simulated cockpit as he views the target on a computer generated display. Fire control computer and software are real, but the aircraft dynamics and the target display must be simulated with the proper time relationships. In the AVSAIL laboratory, the DEC-10 computer provides the simulated environment by executing models of that environment. The interface between the DEC-10 and the system under test is provided by multiple DEC PDP-11 minicomputers that allow more than one system simulation, or complex multicomputer simulations, to be connected simultaneously to the DEC-10. As shown in Figure 1.3.1.1-1, there are currently eight PDP-11's interfaced to the DEC-10. Four of these are devoted to various aspects of the DAIS program, one is devoted to interfacing the F-16 Fire Control Computer Simulator, one to the cockpit simulator currently being used with DAIS, one to the Evans and Sutherland Picture System, and the eighth to the video center.

In order to interface multiple PDP-11's to the DEC-10, hardware and software features have been provided to allow the PDP-11's to transmit and receive data from the DEC-10. The basic mechanism for data communication is a 4K word memory "window" in the DEC-10 memory. The location of the window can be specified by any DEC-10 program and addressed by the PDP-11 computer. Provision for an 18-bit address has been made by hardware modifications to the DMA-10C Direct Memory Access Controller. The memory "window" is formatted to provide storage of "to" and "from" data and window identifier information. Data are stored in the window at each simulation frame or upon interrupt from a PDP-11. Data transfer to and from the PDP-11 is under PDP-11 control.

A special software program, resident in the DEC-10, handles data formatting as required by the different word sizes of the two computers (i.e., 36 bits for the DEC-10 and 18 bits for the PDP-11). This software also provides the functions of interrupt handling from the PDP-11's and the real-time clock.

2.1.1.3 PDP-11 Satellite Computers

Eight Digital Equipment Corporation (DEC) PDP-11 computers are employed within the DEC-10 Host Simulation Processor Hardware in order to provide an interface for each of eight system simulators with the Host Hardware. Each interface is operated in a pseudo

real-time asynchronous interrupt mode via a Direct Memory Interface Controller. The output of the memory controller is coupled to an MX-10 Memory Multiplexer prior to connection to the DEC-10 memory bus. The speed of the multiplexer and direct memory access bus operation is fast enough that each satellite computer appears to be operating in real time to the user. Each system simulator computer can in this manner provide data to common memory blocks of the DEC-10 system. This data can then be shared throughout the facility. The assignment of simulator processors is given in Table 2.1.1.3-1.

The PDP-11 series of computers are well suited to use in a simulation facility primarily due to their single or UNIBUS structure. The UNIBUS is a single, high-speed, bidirectional, asynchronous communications path within each of the PDP-11 computers. It allows all system components and peripheral devices to communicate directly without central processor intervention. This direct communications means that the PDP-11 does not require I/O instructions. The same instruction that performs a register-to-register transfer within the central processor performs:

1. a memory-to-device-register transfer,
2. a memory-to-memory transfer,
3. a device-register-to-memory transfer, and
4. a device-register-to-another-device-register transfer.

Therefore, the key point is that a peripheral device can be communicating with memory at the same time the processor is performing computational operations.

All PDP-11 system elements connect directly to the UNIBUS in plug-in fashion. The asynchronous nature of the UNIBUS means external devices can be tied into the system without regard for individual operating speed. The UNIBUS permits system expansion to any level without revision of the present system. Also, as the full UNIBUS technique is

TABLE 2.1.1.3-1. PDP-11 PROCESSOR ASSIGNMENTS

System Simulator	PDP Computer	System Simulator	PDP Computer
a) DAIS (GT-44A, Figure 2.1.1.3-1)	11/40	e) F16 Simulator	11/40
b) DAIS (GT-44A, Figure 2.1.1.3-1)	11/40	f) Cockpit	11/45
c) DAIS (GT-44B, Figure 2.1.1.3-2)	11/40	g) Evans and Sutherland Picture System	11/50
d) DAIS (GT-44C, Figure 2.1.1.3-3)	11/40	h) Video Center	11/20

common to all PDP-11 systems, peripheral devices can be freely interchanged from system to system without the need for special interfaces.

All PDP-11 processors use the same basic instruction set. Programs developed on the PDP-11/40 of DAIS are immediately usable on the PDP-11/50 of the Picture System. Common software incurs no conversion problems as needs increase. Programs developed on any PDP-11 will serve all anticipated systems, regardless of the specific model. A processor comparison table for the 11/40, 11/45, 11/50, and 11/20 is given in Table 2.1.1.3-2.

2.1.1.3.1 The DAIS simulators

The four PDP 11/40 computers associated with DAIS are employed as shown in Figures 2.1.1.3-1, -2, and -3. Two of the simulators have configuration A shown in Figure 2.1.1.3-1.

TABLE 2.1.1.3-2. PDP-11 PROCESSOR COMPARISON TABLE

Processor Type	11/20	11/40	11/45	11/50
Stack processing	Yes	Yes	Yes	Yes
Programmable stack limit	No	Optional	Yes	Yes
General registers	8	8	16	16
Reg-to-reg. transfer	2.3 μs	900 ns	300 ns	300 ns
Hardware floating point	No	32 bit (opt)	32, 64 bit (opt)	32, 64 bit (opt)
Max memory size (bytes)	56K	248K	248K	248K
Memory type	Core	Core	BIPOLAR*	BIPOLAR
			MOS*	MOS
			Core	Core
Effective memory speed	980 ns	1000 ns	300 ns	300 ns
			500 ns	500 ns
			1000 ns	1000 ns
Memory parity	Optional	Yes	Yes	Yes
Memory management	No	Optional	Yes	Yes
Processing modes	1	2 (opt)	3	3
Auto hardware interrupts	Yes	Yes	Yes	Yes
Auto software interrupts	Yes**	No	Yes	Yes
Power fail/auto restart	Yes	Yes	Yes	Yes
Real-time clock	Optional	Optional	Yes	Yes
Programmer's console	Yes	Yes	Yes	Yes
Hardware bootstrap	Optional	Optional	Yes	Yes
Serial line controller	Yes	Yes	Yes	Yes

*A PDP 11/45 used with BIPOLAR and/or MOS memory becomes the equivalent of a PDP 11/50.

**Automatic interrupts are possible through the use of software TRAP programming.

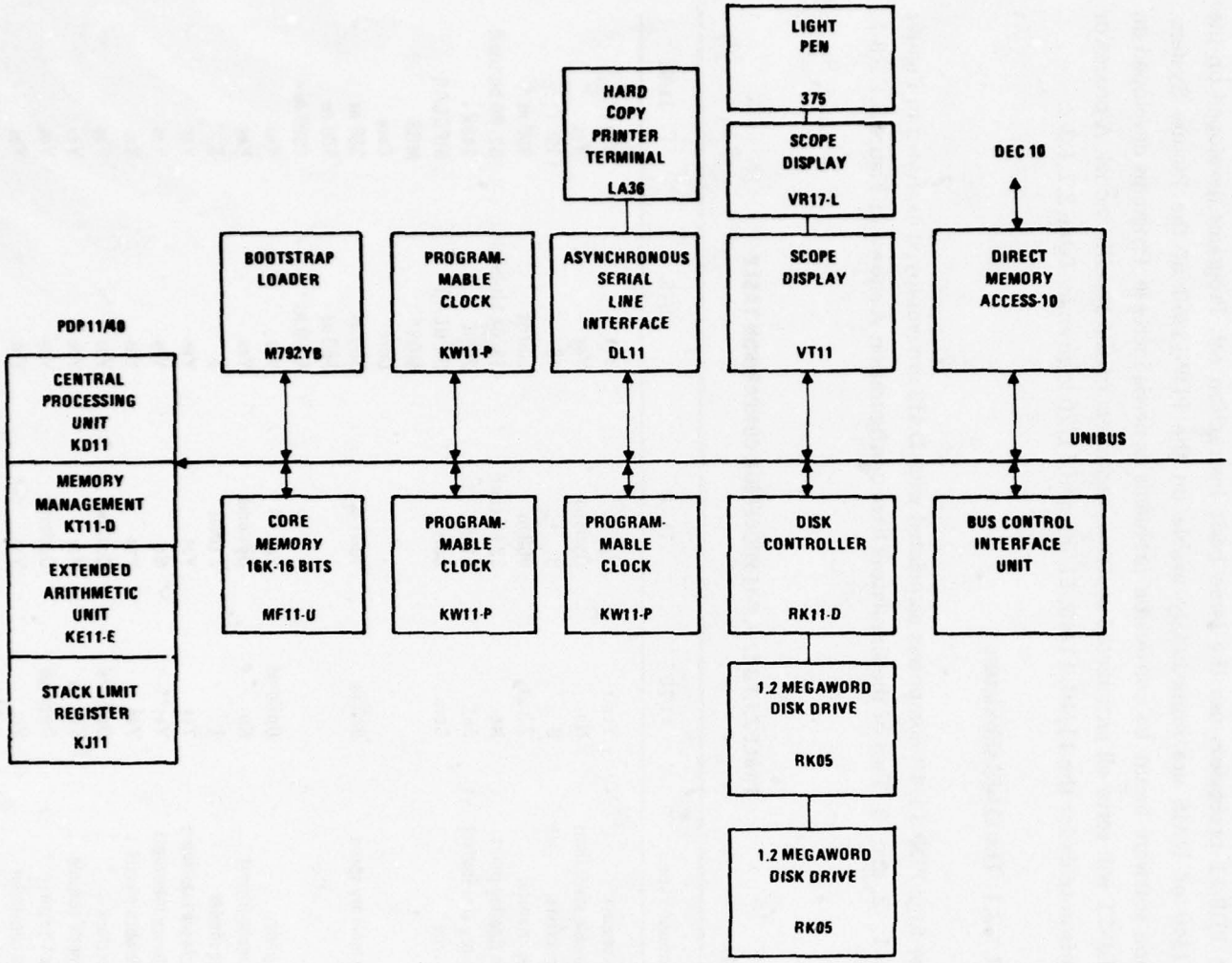


Figure 2.1.1.3-1. DAIS simulator (GT-44A).

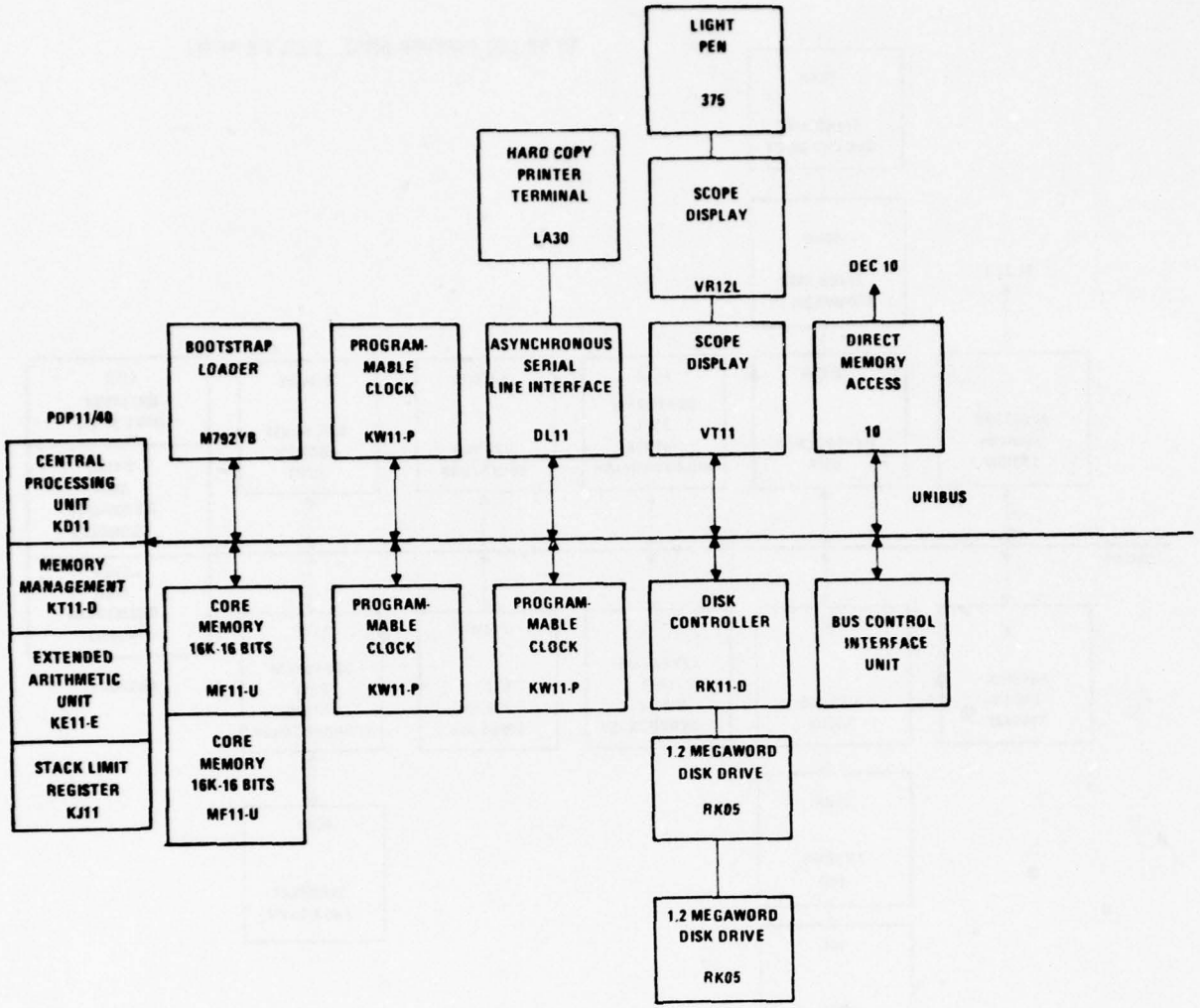


Figure 2.1.1.3-2. DAIS simulator (GT-44 B).

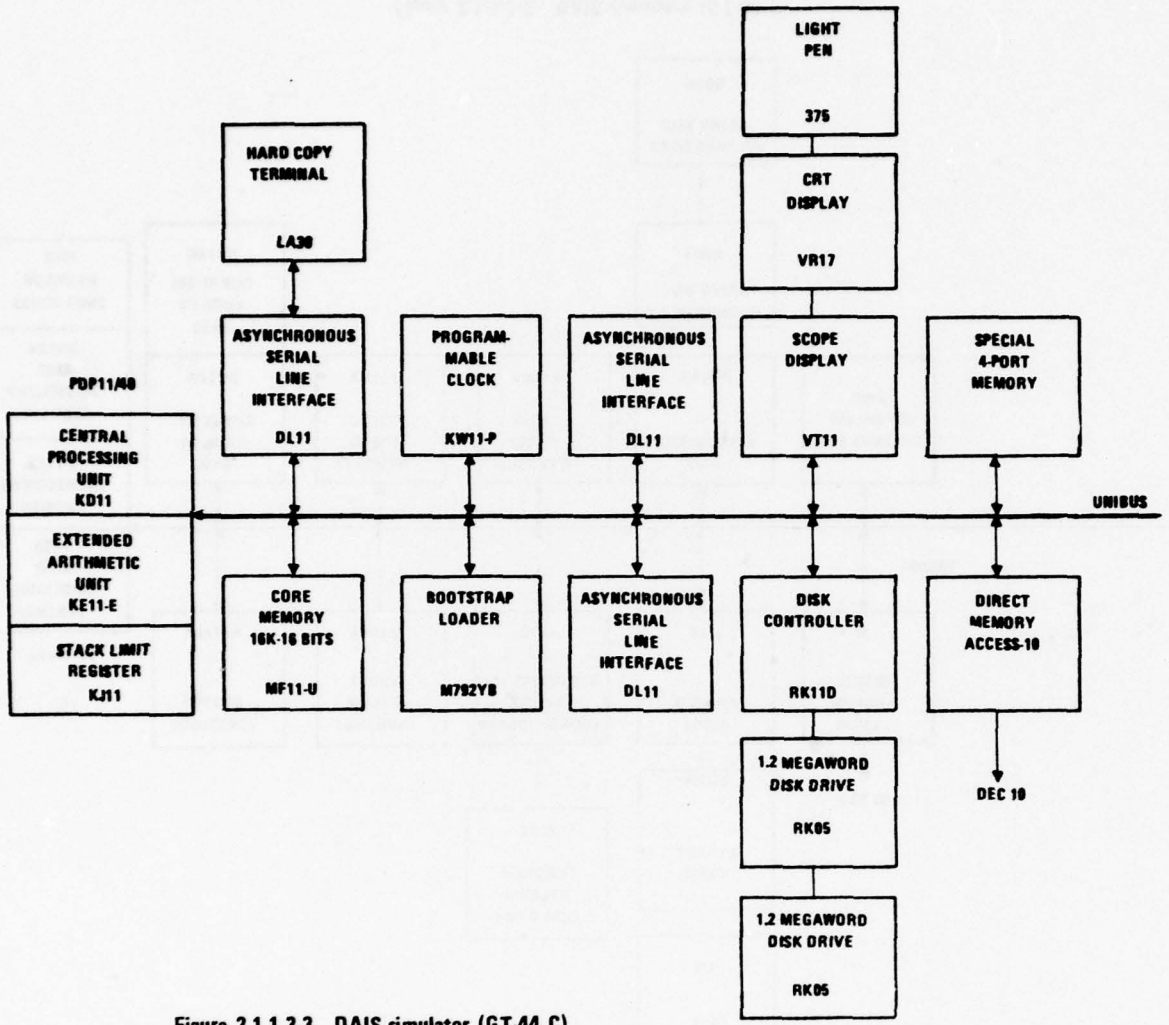


Figure 2.1.1.3-3. DAIS simulator (GT-44 C).

This configuration utilizes 16K of 16-bit core memory as well as 2 1.2 megaword disk memory systems. Configuration B shown in Figure 2.1.1.3-2 employs 32K of 16-bit core memory, 2 1.2 megaword disk drives and a bootstrap loader memory. Configuration C shown in Figure 2.1.1.3-3 contains only 16K of 16-bit core memory, however, it also utilizes 2 1.2 megaword disk drives.

2.1.1.3.2 The F16 simulator

A PDP 11/40 computer is used with the F16 simulator as shown in Figure 2.1.1.3-4. The basic memory provided for this configuration consists of a 24K word by 16-bit core memory, and a 1.2 megaword disk memory system. The 11/40 is used in this simulator in order to control the Fire Control Navigation Panel, and Fire Control Computer simulator. It also receives Flight Control Stick and Throttle data as inputs and provides control information to a local Head Up Display CRT.

2.1.1.3.3 The cockpit simulator

The cockpit simulator utilizes a PDP 11/45 as shown in Figure 2.1.1.3-5. In this configuration a high-speed 28K by 16-bit MOS memory as well as a 68K word by 16-bit core memory are used. The 11/45 processor provides control and data for the Horizontal Situation Display and Vertical Situation Display as well as the control stick and associated lamp and keyboard displays. These controls and displays are all located within the simulated cockpit.

2.1.1.3.4 The Picture System

The Picture System may be interfaced and controlled by any PDP-11 computer. Picture Systems have been interfaced to PDP-11/05, PDP-11/35 and PDP-11/45 computers with various standard DEC peripherals including disks, DECtapes, megtapes, printers, etc. The standard Refresh Buffer requirements is 8K of 36-bit words. An additional 8K of Refresh Memory may be obtained to provide a 16K Refresh Buffer.

The Picture System currently employs a PDP-11/50 as shown in Figure 2.1.1.3-6. In this configuration, two core memories of 16K and 44K each are utilized as well as 32K of MOS memory.

2.1.1.3.5 The Video Center

The Video Center currently employs a PDP-11/20 as shown in Figure 2.1.1.3-7. This

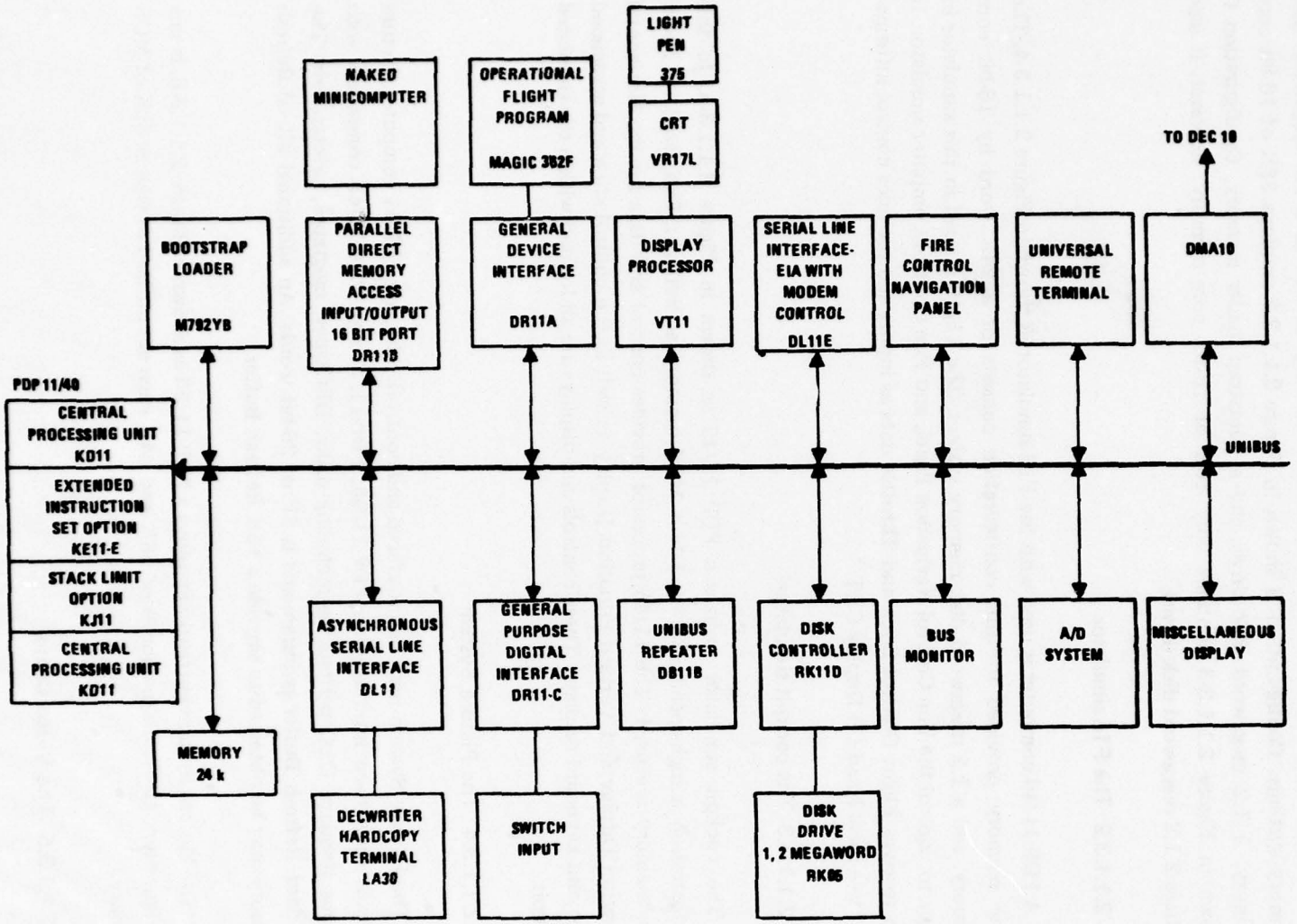


Figure 2.1.1.3-4. F-16 simulator.

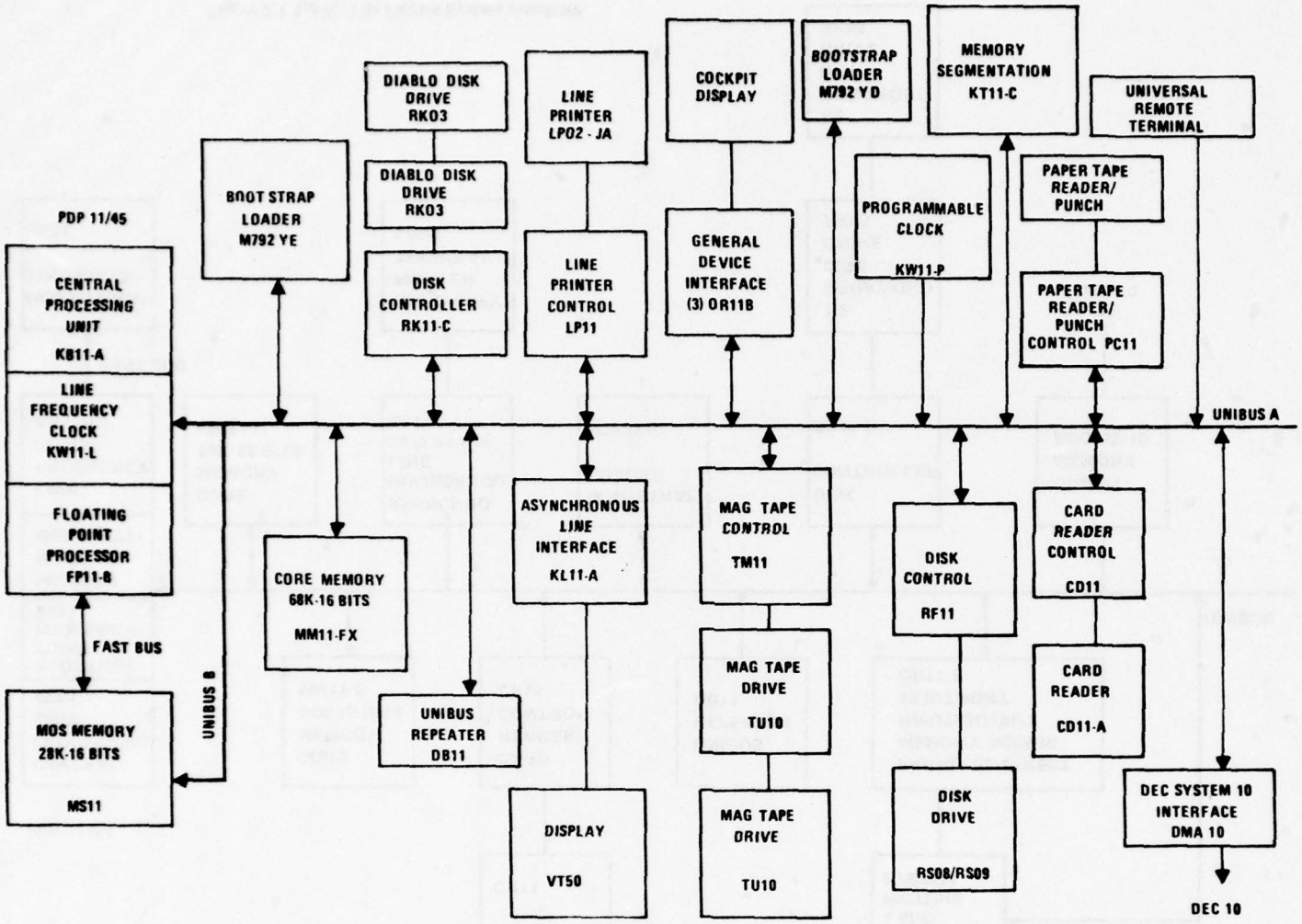


Figure 2.1.1.3-5. The cockpit simulator.

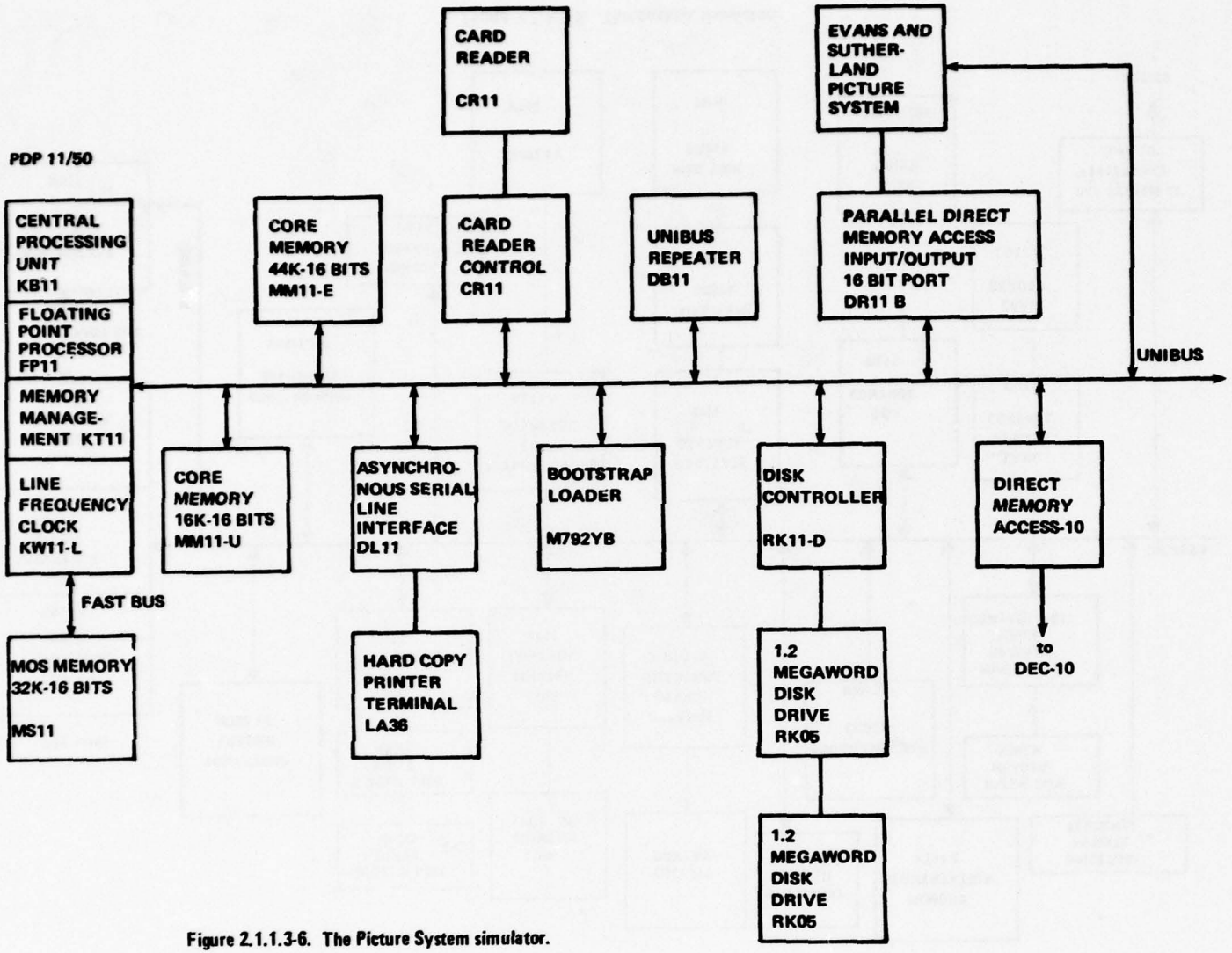


Figure 2.1.1.3-6. The Picture System simulator.

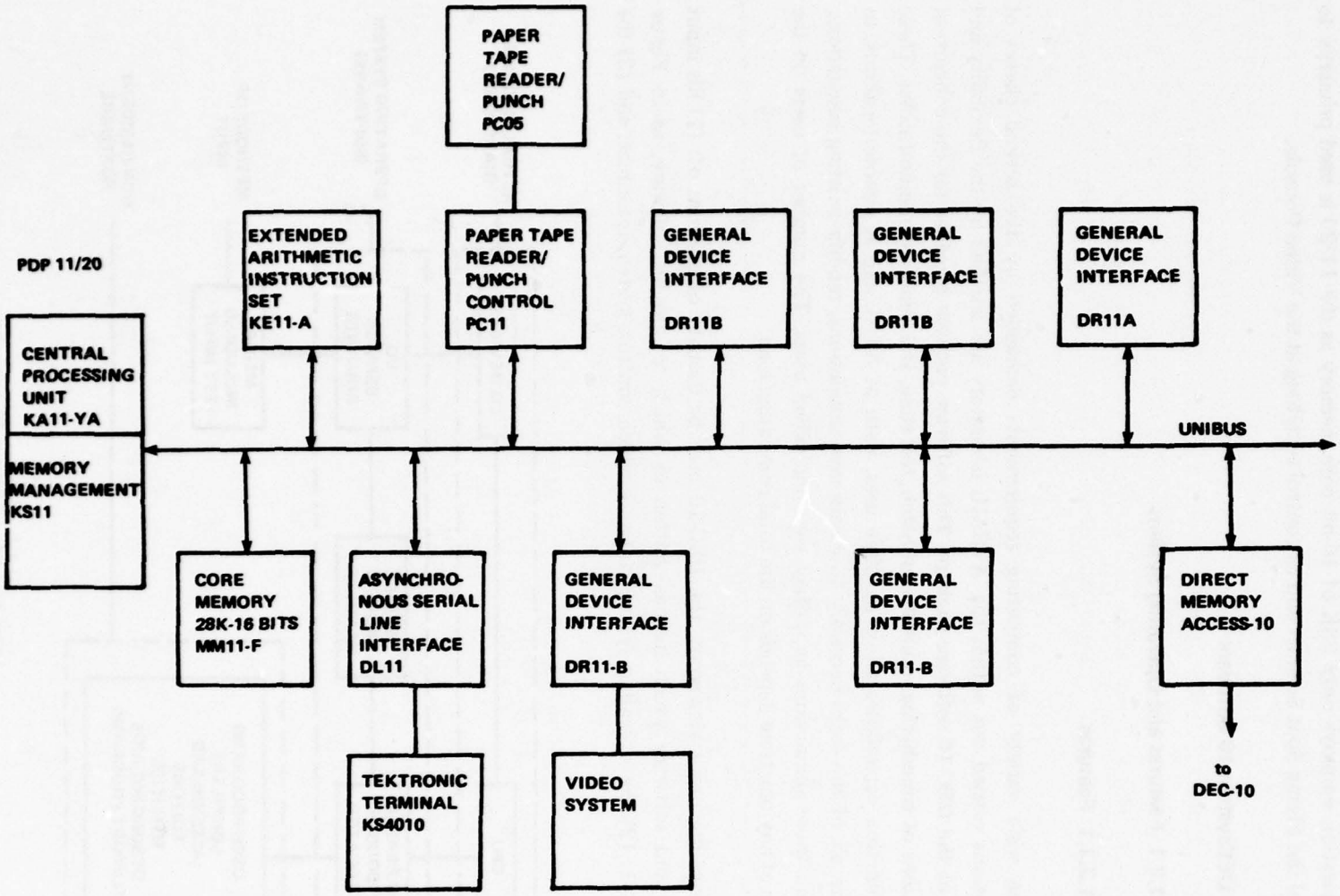


Figure 2.1.1.3-7. The Video Center simulator.

configuration employs only 28K of 16-bit core memory as the 11/20 is used primarily to control the Flying Spot Scanner and to control switching at the Video Console.

2.1.2 DECsystem-10 Software

2.1.2.1 Features and Operating System

2.1.2.1.1 Features

The wide variety of computing requirements demanded by the several classes of simulations carried out within the AVSAIL laboratory are satisfied by the flexibility and scope of the DEC-10 software package. This software package provides for the concurrent operations of timesharing, multistream batch, real time, and remote communications. These multifunction capabilities allow multiple uses, both at AFAL and at remote locations, to perform all of the tasks necessary to create new simulations, modify existing simulations, and run those simulations as if they were individual users. The number of users on the system at any one time depends on the total computing load.

From the user's viewpoint, the DEC-10 may be thought of in terms of: (1) his input device and software which he has written or which act on his software, as in Figure 2.1.2.1-1; (2) the operating system software which controls system resources; and (3) the

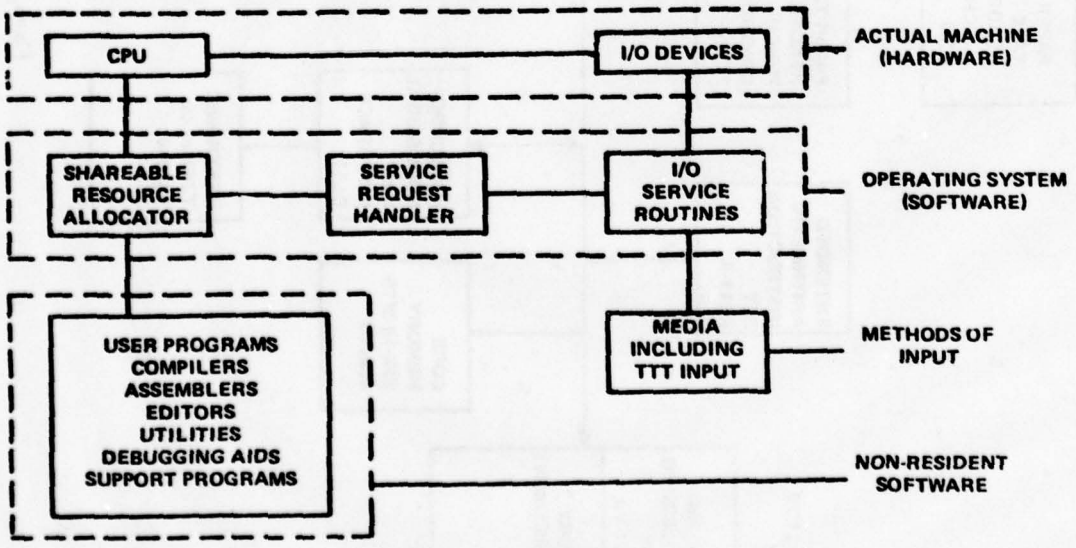


Figure 2.1.2.1-1. DECsystem-10, user's view.

system hardware which was previously described. The DEC-10 has several capabilities which increase the utilization of system resources in a multiuser environment. These are described in the following three sections.

2.1.2.1.1.1 Timesharing. The timesharing capability allows resources to be shared among users. The timesharing environment utilizes processor time and system resources that are wasted in single-user systems. Users are not restricted to a small set of system resources, but instead are provided with the full variety of facilities. By means of his terminal, the user has online access to most of the system's features. This online access is available through the operating system command control language, which is the means by which the timesharing user communicates with the system.

Through the command language, the user controls the running of a task, or job, to achieve the desired results: create, edit, and delete files; start, suspend, and terminate a job; compile, execute, and debug a program. In addition, since multiprogramming batch software accepts the same command language as the timesharing software, any user can enter a program into the batch run queue. Thus, any timesharing terminal can act as a remote job-entry terminal.

With the command language, the user can also request assignment of any peripheral device (magnetic tape, DECtape, and private disk pack) for exclusive use. When the request for assignment is received, the operating system verifies that the device is available to this user, and the user is granted its private use until he relinquishes it. In this way, the user can also have complete control of devices such as card readers and punches, paper tape readers and punches, and line printers.

When private assignment of a slow-speed device (card punch, line printer, paper tape punch, and plotter) is not required, the user can employ the spooling feature of the operating system. Spooling is a method by which output to a slow-speed device is placed on a high-speed disk or drum. This technique prevents the user from consuming unnecessary time and space in core while waiting for either a device to become available or output to be completed. In addition, the device is managed to a better degree because the users cannot tie it up indefinitely, and the demand fluctuations experienced by these devices are equalized.

2.1.2.1.1.2 Multiprogramming. The DEC-10 has the capability to make maximum utilization of memory. The DEC-10 is a multiprogramming system; i.e., it allows multiple independent-user programs to reside simultaneously in memory and to run concurrently. This technique of sharing memory and processor time enhances the efficient operation of the system by switching the processor from a program that is temporarily stopped because

of I/O transmission to a program that is executable. When core and the processor are shared in this manner, each user's program has a memory area distinct from the area of other users. Any attempt to read or change information outside of the area a user can access immediately stops the program and notifies the operating system. Because available memory can contain only a finite number of programs at any one time, the computing system employs a secondary memory, usually disk or drum, to increase the number of users serviced. User programs exist on the secondary memory and move into memory for execution. Programs in memory exchange places with the programs being transferred from secondary memory for maximum use available main memory. Because the transferring, or swapping, takes place directly between main memory and the secondary memory, the central processor can be operating on a user program in one part of memory while swapping is taking place in another. This independent, overlapped operation greatly improves system utilization by increasing the number of users that can be accommodated at the same time.

To further increase the utilization of memory, the operating system allows users to share the same copy of a program or data segment. This prevents the excessive memory usage that results when a program is duplicated for several users. A program that can be shared is called a reentrant program and is divided into two parts or segments. One segment contains the code that is not modified during execution (e.g., compilers and assemblers) and can be used by any number of users. The other segment contains nonreentrant code and data. The operating system provides for shared segments to guarantee that they are not accidentally modified.

2.1.2.1.1.3 File Protection. The DEC-10 has the capability to manage the storage of user program and data files consistent with the multiple-user environment. The mass storage devices available are shared among users, and thus, the operating system must ensure independence among the users; one user's actions must not affect the activities of another unless the users desire to work together. To guarantee such independence, the operating system provides a file system for disks, disk packs, and drums. Each user's data is organized into groups of 128-word blocks called files. The user gives a name to each of his files, and the list of these names is kept by the operating system for each user. The operating system is then responsible for protecting each user's file storage from intrusion by unauthorized users. The operating system lets the user specify protection rights, or codes, for his files. These codes designate if others may read the file, and after access, if the files can be modified in any way. Files are assigned protection levels for each three classes of users: self, users with a common project number, and all users. Each user class may be assigned a different access privilege, so that there are eight levels in each of the three user classes as described by Table 2.1.2.1-1. This file protection scheme results in a three-digit access code for all files.

TABLE 2.1.2.1-1. FILE PROTECTION CODES

Protection Level	Access Code	Access Privileges
Greatest protection	7	No access privileges
	6	EXECUTE ONLY
	5	READ, EXECUTE
	4	APPEND, READ, EXECUTE
	3	UPDATE, APPEND, READ EXECUTE
	2	WRITE, UPDATE, APPEND, READ, EXECUTE
	1	RENAME, WRITE, UPDATE, APPEND, READ, EXECUTE
Least protection	0	CHANGE POSITION, RENAME, WRITE, UPDATE, APPEND, READ, EXECUTE

2.1.2.1.2 Operating system

In order to have some better appreciation for the manner in which the resources of the DEC-10 are managed, it is helpful to examine the nature of the operating system alluded to by Figure 2.1.2.1-1. The resident operating system is made up of a number of separate and somewhat independent parts, or routines (Figure 2.1.2.1-2). Some of these routines are cyclic in nature and are repeated at every system clock interrupt (tick) to ensure that every user of the computing system is receiving the requested services. These cyclic routines are:

1. The command processor, or decoder,
2. The scheduler,
3. The swapper.

The command decoder is responsible for interpreting commands typed by the user on his terminal and passing them to the appropriate system program or routine. The scheduler decides which user is to run in the interval between the clock interrupts, allocates sharable system resources, and saves and restores conditions needed to start a program interrupted by the clock. The swapper rotates user jobs between secondary disk memory and core memory after deciding which jobs should be in core but are not. These routines constitute the part of the operating system that allows many jobs to be operating simultaneously.

The noncyclic routines of the operating system are invoked only by user programs and

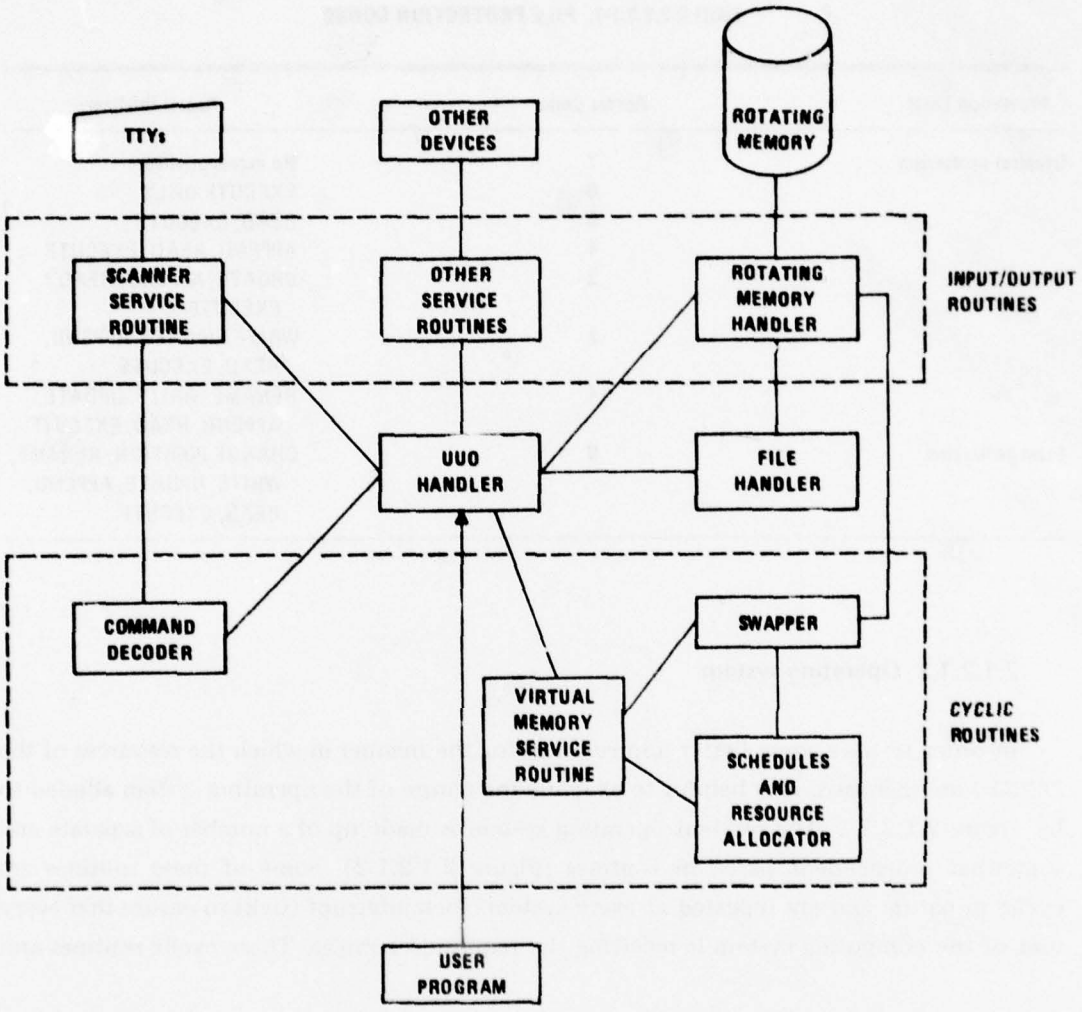


Figure 2.1.2.1-2. The resident operating system.

are responsible for providing these programs with the services available through the operating system. These routines are:

1. The Unimplemented User Option (UUO),
2. The input/output routines,
3. The file handler.

The UUO handler is the means by which the user program communicates with the operating system in order to have a service performed. Communication is by way of programmed operators (also known as UUO's) contained in the user program which, when executed, go to the operating system for processing. The input/output routines are the routines responsible for directing data transfers between peripheral devices and user

programs in core memory. These routines are invoked through the UUO handler, thus saving the user the detailed programming needed to control peripheral devices. The file handler adds permanent user storage to the computing system by allowing users to store named programs and data as files.

2.1.2.1.2.1 Command Decoder. The Command Decoder is the communications link between the user's terminal and the operating system. Because all the requests for system resources are initiated via the command decoder, it is the most visible part of the system to each user. When the user types commands and/or requests on his terminal, the characters are stored in an input buffer in the operating system. The command decoder examines these characters in the buffer, checks them for correct syntax, and invokes the system program or user program as specified by the command.

On each clock interrupt, control is given to the command decoder to interpret and process one command in the input buffer. Given that the command is a legal one, several actions are possible. For instance, a command must be delayed if the job is swapped out to the disk and the command requires that the job be resident in core; the command is executed on a later clock interrupt when the job is back in core. If all conditions as specified by the legality flags are met, control is passed to the appropriate program.

2.1.2.1.2.2 Scheduler. The DEC-10 is a multiprogramming system; i.e., it allows several user jobs to reside in core simultaneously and to operate sequentially. It is then the job of the scheduler to decide which jobs should run at any given time. In addition to the multiprogramming feature, the DEC-10 employs a swapping technique whereby jobs can exist on an external storage device (e.g., disk or drum) as well as in core. Therefore, the scheduler decides not only what job is to be run next, but also when a job is to be swapped out onto disk or drum and later brought back into core.

All jobs in the system are retained in ordered groupings called queues. These queues have various priorities that reflect the status of each job at any given moment. The queue in which a job is placed depends on the system resource for which it is waiting and, because a job can wait for only one resource at a time, it can be in only one queue at a time. Several of the possible queues in the system are:

1. Run queues for jobs waiting for, or jobs in, execution,
2. I/O wait queues for jobs waiting for data transfers to be completed,
3. I/O wait-satisfied queues for jobs waiting to run after data transfers have been completed,

4. Resource wait queues for jobs waiting for some system resource,
5. Null queue for all job numbers that are not currently being used.

The job's position within certain queues determines the priority of the job with respect to other jobs in the same queue. For example, if a job is first in the queue for a sharable device, it has the highest priority for the device when it becomes available. However, if a job is in an I/O wait queue, it remains in the queue until the I/O is completed. Therefore, in an I/O wait queue, the job's position has no significance. The status of a job is changed each time it is placed into a different queue.

In additon, data transfers use the scheduler to permit the user to overlap computation with data transmission. In unbuffered data modes, the user supplies an address of a command list containing pointers to locations in his area to and from which data is to be transferred. When the transfer is initiated, the job is scheduled into an I/O wait queue where it remains until the device signals the scheduler that the entire transfer has been completed.

In buffered modes, each buffer contains information to prevent the user and the device from using the same buffer at the same time. If the user requires the buffer currently being used by the device as his next buffer, the user's job is scheduled into an I/O wait queue. When the device finishes using the buffer, the device calls the scheduler to reactivate the job.

2.1.2.1.2.3 Swapper. The swapper is responsible for keeping in core the jobs most likely to be run. It determines if a job should be in core by scanning the various queues in which a job may be. If the swapper decides that a job should be brought into core, it may have to take another job already in core and transfer it to secondary memory. Therefore, the swapper is not only responsible for bringing a job into core but also responsible for selecting the job to be swapped out. The swapper periodically checks to see if a job should be swapped in. If there is no such job, then it checks to see if a job is requesting more core. If there is no job wishing to expand its size, then the swapper does nothing further and relinquishes control of the processor until the next clock tick.

2.1.2.1.2.4 UUO Handler. The UUO handler is responsible for accepting requests for services available through the operating system. These requests are made by the user program via software-implemented instructions known as programmed operators, or UUOs. The UUO handler is the only means by which a user program can give control to the operating system in order to have a service performed. The user informs the operating system of his requirements for I/O by means of UUO's contained in his program. The actual input/output routines needed are then called by the UUO handler.

2.1.2.1.2.5 Input/Output. Since the operating system channels communication between the user and the device, the user does not need to know all the peculiarities of each device on the system. In fact, the user program can be written in a similar manner for all devices. The operating system will ignore, without returning an error message, operations that are not pertinent to the device being used. Thus, a terminal and a disk file can be processed identically by the user program. In addition, user programs can be written to be independent of any particular device. The operating system allows the user program to specify a logical device name, which can be associated with any physical device at the time when the program is to be executed. Because of this feature, a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. The device can be designated as a logical device name and assigned to an available physical device with one command to the operating system.

2.1.2.1.2.6 File Handler. The disk file handler manages user and system data; thus, this data can be stored, retrieved, protected, and/or shared among other users of the computing system. All information in the system is stored as named files in a uniform and consistent fashion, thus allowing the information to be accessed by name instead of by physical disk addresses. Therefore, to reference a file, the user does not need to know where the file is physically located. A named file is uniquely identified in the system by a file name and extension, an ordered list of directory names (UFDs and SFDs) which identify the owner of the file, and a file structure name which identifies the group of disk units containing the file.

Usually a complete disk system is composed of many disk units of the same and/or different types. Therefore, the disk system consists of one or more file structures—a logical arrangement of files on one or more disk units of the same type. This method of file storage allows the user to designate which disk unit of the file structure he wishes to use when storing files. Each file structure is logically complete and is the smallest section of file memory that can be removed from the system without disturbing other units in other file structures. All pointers to areas in a file structure are by way of logical block numbers rather than physical disk addresses; there are no pointers to areas in other file structures, thereby allowing the file structure to be removed.

All disk files are composed of two parts, data and information used to retrieve data. The retrieval part of the file contains the pointers to the entire file, and is stored in two distinct locations on the device and accessed separately from the data. System reliability is increased with this method because the probability of destroying the retrieval information is reduced; system performance is improved because the number of positionings needed for random-access methods is reduced. The storing of retrieval information is the same for both

sequential and random-access files. Thus a file can be created sequentially and later read randomly, or vice versa, without any data conversion.

To the user, a file structure is like a device; i.e., a file structure name or a set of file structure names can be used as the device name in command strings or UUO calls to the operating system. Although file structures or the units composing the file structures can be specified by their actual names, most users specify a general, or generic, name (DSK) which will cause the operating system to select the appropriate file structure. The appropriate file structure is determined by a job search list. Each job has its own job search list with the file structure names in the order in which they are to be accessed when the generic name is specified as the device. This search list is established by LOGIN and thus each user has a UFD for his project-programmer number in each file structure in which LOGIN allows him to have files. File storage is dynamically allocated by the file handler during program operations, so the user does not need to give initial estimates of file length or the number of files.

2.1.2.1.3 Real-time operating system features

The multiple simulators attached to the DEC-10 (Figure 1.3.1.1-1) which require software simulations to be carried out by the DEC-10 in real time, as well as handle the attendant data transfer between the DEC-10 and the PDP-11s, call upon the real-time capabilities of the operating system. The operating system must allocate system resources dynamically in order to satisfy the response and computational requirements of real time without affecting the simultaneous operations of timesharing and batch jobs. As part of its normal operation, the DEC-10 operating system satisfies this response requirement by overlapping I/O operations with processing time and by reacting to a constantly changing system load.

At the same time, each user of the computing system must be protected from other users, just as the system itself is protected from all user program errors. In addition, since real-time systems have special real-time devices associated with jobs, the computing system must be protected from hardware faults that could cause system breakdown. And, because protection is part of the function of the operating system, the real-time software employs this feature to protect users as well as itself against hardware and software failures. Inherent in the operating system is the capability of real time, and it is by way of calls to the operating system that the user obtains real-time services. The services obtained by calls within the user's program include:

1. Locking a job in core,

2. Connecting a real-time device to the priority interrupt system,
3. Placing a job in a high-priority run queue,
4. Initiating the execution of FORTRAN or machine language code on receipt of an interrupt,
5. Disconnecting a real-time device from the priority interrupt system.

Memory space may be occupied by the resident operating system and by a mix of real-time and non-real-time jobs. The only fixed partition is between the resident operating system and the remainder of memory. A real-time job may need to be in memory so as not to lose information when its associated real-time device interrupts (since there may not be sufficient time to swap-in the job). The job can request that it be locked into core.

The real-time user can connect real-time devices to the priority interrupt system, respond to these devices at interrupt level, remove the devices from the interrupt system, and/or change the priority interrupt level on which these devices are assigned. There is no requirement that these devices be connected at system generation time. The user specifies both the names of the devices generating the interrupts and the priority levels on which the devices function. The operating system then links the devices to the operating system.

The real-time user can receive faster response by placing jobs in high-priority run queues. These queues are examined before all other queues in the computing system, and any runnable job in high-priority queue is executed before jobs in other queues. In addition, jobs in high-priority queues are not swapped to secondary memory until all other queues have been scanned.

2.1.2.1.4 Remote communications

The DEC-10 allows the simultaneous operation of multiple remote stations. Software provisions differentiate one remote station from another. By utilizing peripheral devices at various stations, the user is provided with increased capabilities as shown in Figure 2.1.2.1-3. For example, data can be collected from various remote stations, compiled and processed at the central station, and then the results of the processing can be sent to all contributors of the data.

2.1.2.1.5 Batch computing

In addition to the timesharing and real-time capabilities available on the DEC-10, batch computing is provided by the GALAXY-10 batch software. GALAXY-10 batch software enables the DEC-10 to execute up to 128 batch jobs concurrently with timesharing jobs.

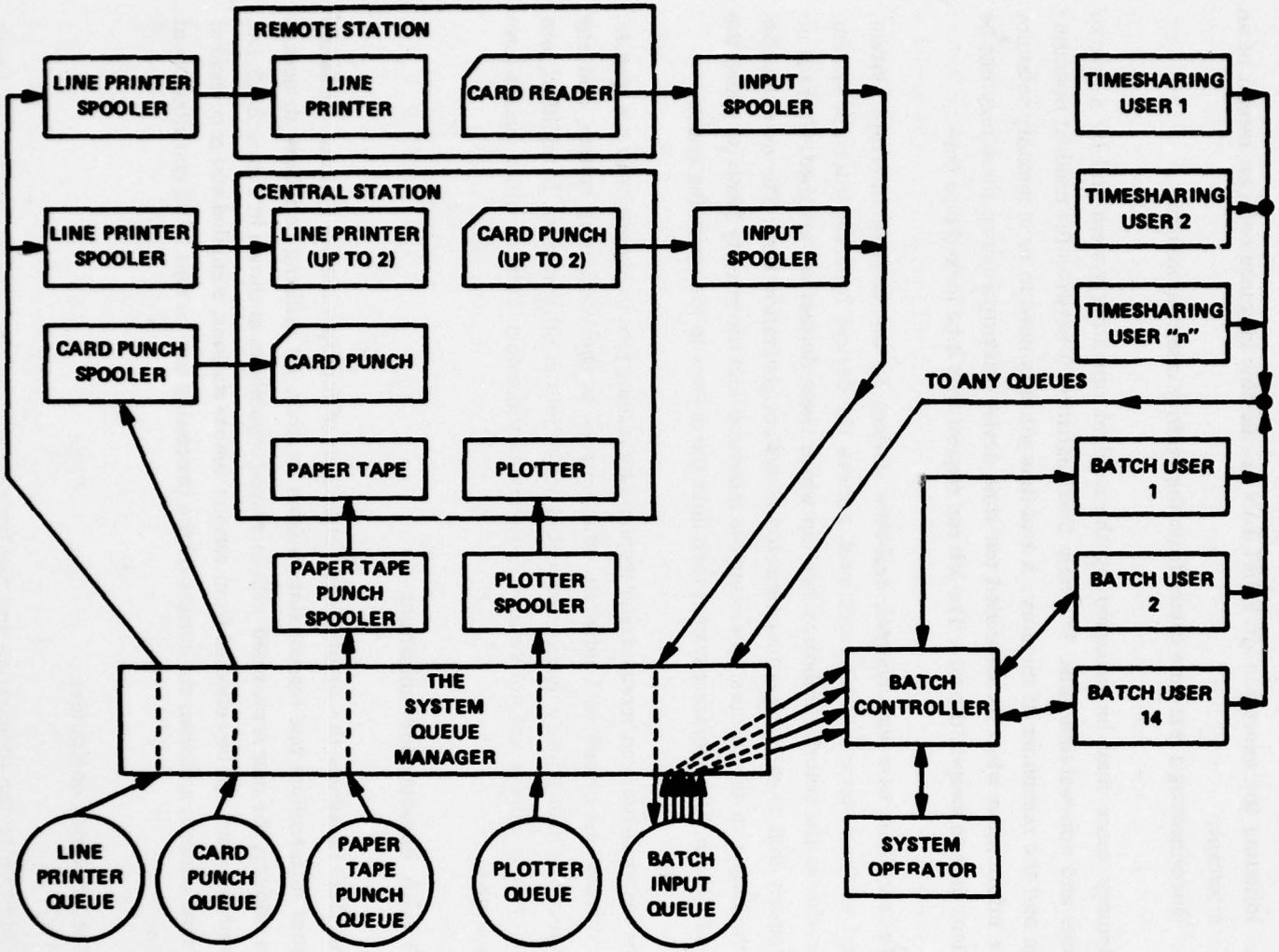


Figure 2.1.2.1-3. Remote communications.

Just as the timesharing user communicates with the system by way of his terminal, the batch user normally communicates by way of the card reader. (However, he can also enter his job from an interactive terminal.) Unlike the timesharing user, the batch user can punch his job on cards, insert a few appropriate control cards, and leave the job for an operator to run. In addition, the user can debug the program in the timesharing environment and then run it in batch mode without any additonal coding.

The GALAXY-10 system consists of a series of programs: QUASAR, the system queue manager and scheduler; SPRINT, the input spooler; BATCON, the batch controller; and the output spoolers, LPTSPL and SPROUT. The input spooler is responsible for reading the input from the input device and for entering the job into the batch controller's input queue. After the input spooler reads the end-of-file and closes the disk files, it makes an entry in the batch controller's input queue. The batch controller processes batch jobs by reading the entries in its queue. The control file created by the input spooler is read by the batch controller, and data and nonresident software commands are passed directly to the user's job.

QUASAR is responsible for scheduling jobs and maintaining both the batch controller's input queue and the output spooling queues. A job is scheduled to run under the batch controller according to external priorities, processing time limits, and core requirements which are dynamically computed, and according to parameters specified by the user for his job, such as start and deadline time limits for program execution.

The output spooling programs improve system throughput by allowing the output from a job to be written temporarily on the disk for later transfer, instead of being written immediately on a particular output device. The log file and all job output are placed into one or more output queues to await processing. When the specified device is available, the output is then processed by the appropriate spooling program. These spooling programs may be utilized by all users of the computing system.

2.1.2.2 Program Support Software

The preparation of software programs in both assembly language and in several higher order languages is facilitated by the various utility programs and compilers supported on the AVSAIL DEC-10. The basic features of each of these support programs are briefly summarized below.

2.1.2.2.1 Higher order language compilers

A wide range of higher order language capability is supported by the DEC-10 AVSAIL facility. Languages available include JOVIAL, FORTRAN, APL, Basic, COBOL, and others as subsequently described. The descriptions, other than for JOVIAL, are brief since these languages are widely used and documented.

2.1.2.2.1.1 JOVIAL. The JOVIAL language is the Air Force standard language for command and control software, and the level I subset of J73 is supported with a compiler on the DEC-10.

JOVIAL had its beginnings in 1958 and, as the acronym (Jule's Own Version of the International Algebraic Language) implies, is similar to ALGOL in many ways. Subsequent modifications over the years have left it substantially different, however. The introduction of JOVIAL brought with it a number of innovative software features. It was the first language (and until PL/I, the only one) to provide good facilities for simultaneously performing scientific numerical computation and nontrivial data handling, while at the same time it could also be used in general information handling areas. A second contribution is the use of COMPOOL (COMmunication POOL) as a central source of data description. A third contribution was its practical usage as its own compiler, and finally, it made a significant contribution in terms of allowing the programmer great flexibility for controlling storage allocation when he needs to, but not requiring him to do so otherwise. It is not within the scope of this manual to define the JOVIAL language. Readers familiar with FORTRAN and ALGOL will find many similarities. A few distinctive features of JOVIAL are mentioned here to differentiate it from other higher order languages.

JOVIAL is a procedure-oriented, problem-oriented, and problem-defining language. Its basic objective is to provide a language for use in solving large, complex information processing problems. Possibly the most distinctive feature of JOVIAL related to this basic objective is the COMPOOL. The COMPOOL is a facility which allows for creation of one or more preprocessed common data base descriptions. The COMPOOL source, as defined by a COMPOOL directive and declarations, contains two types of information. First, data declarations which are common to two or more programs may be described in a COMPOOL. In addition, any external procedures or functions may be declared in the COMPOOL. The COMPOOL process involves essentially a compilation of the COMPOOL source, creating two forms of output. One output is a relocatable object module containing space reservation for the data delcared in the COMPOOL and any presets defined for this COMPOOL data. The second output is a special file containing names declared in the COMPOOL and their

attributes for use by the compiler during subsequent compilations which refer to the names declared in the COMPOOL.

Another feature of J73/I worthy of note is the absence of input/output statements. Communication with JOVIAL programs is via the compool data base. Directives are available for accessing auxiliary source files, however.

Certain features of the J73 data declarations are of interest. J73 supports three basic data structures: items (scalar variables); tables of one to seven dimensions; and blocks. Scalar items, tables and blocks may be grouped in blocks. Data may be allocated to one of three levels. The primary and most permanent data is reserve data. Only those values that are left in reserve data upon exit from a procedure are guaranteed at re-entrance to that procedure. Procedure data values are not valid after exit from the containing procedure. The final level of data is based data. No storage is allocated for based data by the compiler. Based data describe a structuring of data, a template, which may be relocated dynamically. This relocation may be performed by declaring a default base—an item whose value is to be used as the address—or at each reference by using a formula whose value is the address. The allocation level to be ascribed to data is indicated by an allocation specifier in the declaration. J73 also provides for packing of table items at three levels: no packing, dense packing, or medium packing. The level of packing implies the extent to which more than one item is stored in a computer word.

The J73/I subset provides two types of statements; statements that compute values and statements that control program flow. Statements may be named or not and may be either simple or compound (delimited by BEGIN - END). Program control statements include GOTO, STOP, RETURN, IF-ELSE, WHILE and a particularly powerful FOR-Loop construct which is, roughly FOR (control variable), BY (increment phrase), THEN (replacement phrase), and WHILE (terminator phrase). The BY, THEN, WHILE elements may appear in any order.

The J73 language is program and procedure-oriented and a procedure call statement is included. The J73 compilation may be a program that is invoked by an operating system or a procedure called from a program or other procedure. Within a compilation (program or procedure), internal procedure declarations may be nested to any level. Within a program, a stop statement is used to terminate a program or procedure and return to the system. A return statement may not be used within a program but may be used within a procedure.

Some additional insight into the nature of JOVIAL is provided by the following brief discussion of the J73/I compiler which is composed of a data base, a set of seven logical

processing phases, and an executive program which supports input/output, phase loading, and commonly used utility functions. Interphase communication is via the global data blocks, tables, and files that comprise the compiler's data base. The structure of the J73/I compiler is presented in Figure 2.1.2.2-1. Descriptions of the compiler components are provided below.

The Compiler Executive (ECEX) is a collective name for those procedures which remain resident throughout a compilation. Resident procedures are of three types:

1. Host Computer Operating System Interface Routines where routines are coded in assembly language; they support phase loading and file input/output and must be completely recoded to rehost the compiler;
2. A collection of conversion and data manipulation procedures, and compiler debugging procedures which output symbolic dumps of compiler files and tables; the conversion and data manipulation procedures are coded in assembly language, and debugging procedures are coded in J73/I;
3. A collection of symbol table service procedures which search and create symbol table entries; these procedures are coded in J73/I.

The Control Card Interpreter (CCI) reads and processes control card command statements for the compilation. Examples of options selectable by a command statement are: target computer identity, input and output file names, and listing options. CCI is currently coded in assembly language.

The Compool Input Processor (CIP) is called after control card interpretation to process compool directives. CIP presets the symbol table with entries from the compool files named in the compool directives. The process consists of reading a compool file's directory, searching for specified entities, obtaining the specified entities from the body of the Compool File, and constructing the appropriate symbol table entries.

CIP is coded in J73/I. Except for certain data declarations, it is host computer independent. CIP and other target independent compiler modules access a global table (TRGPARM) of target parameters, such as bits per word, bits per byte, and address size, in order to generate target-specific output.

Syntax analysis is performed by the Analyzer (ANZR). ANZR translates dynamic statements to Polish postfix form in the Intermediate Language (IL) file, translates declarative statements into symbol table entries, and copies constant presets and cross-reference information to the code file. ANZR is coded in J73/I.

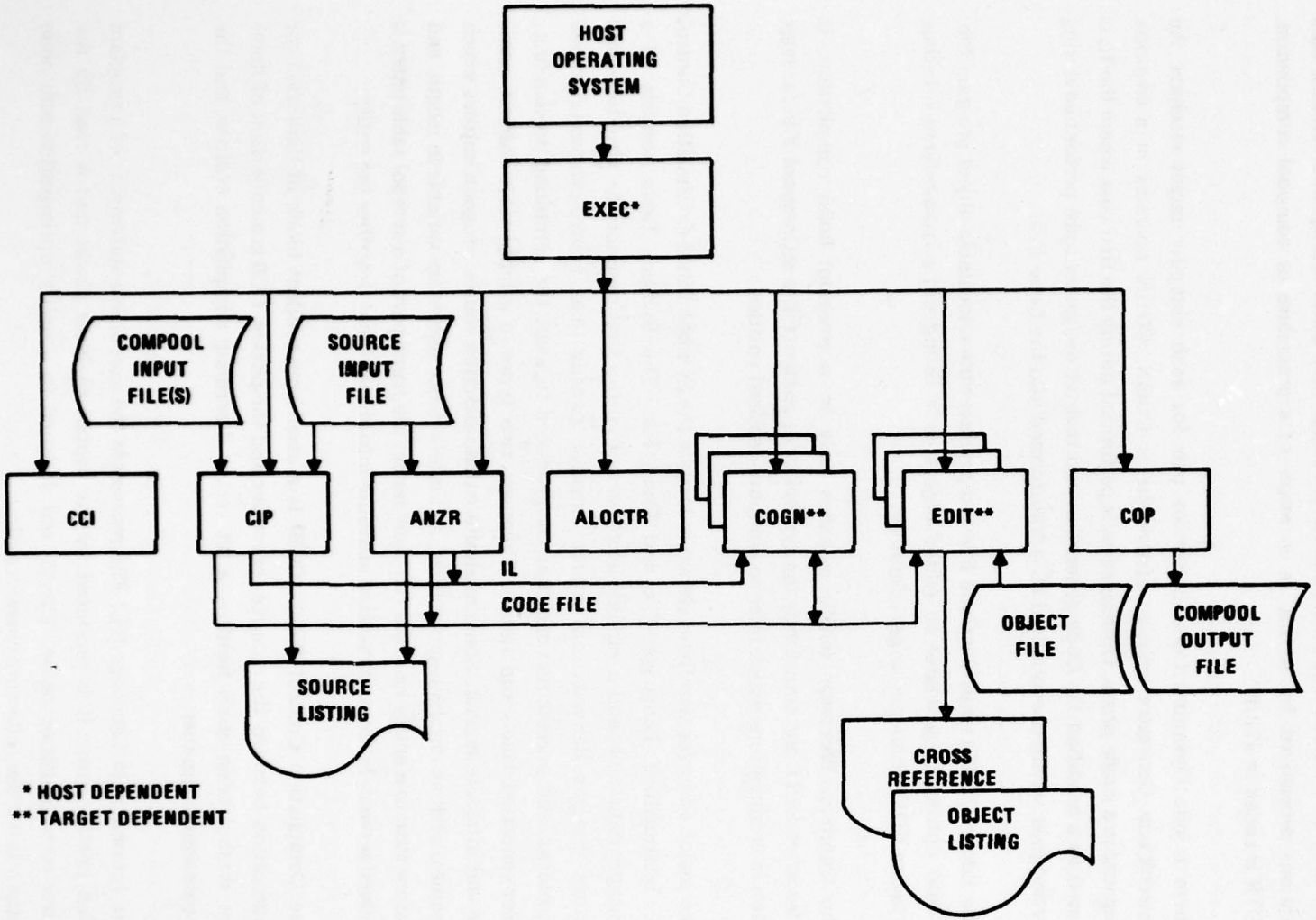


Figure 2.1.2.2-1. J73/I compiler structure.

The Allocator (ALOCTR) is responsible for assigning relative storage addresses for data declarations recognized by ANZR as as result of a procedure or compool compilation. ALOCTR is coded in J73/I.

There is one Optimizer/Code Generator pair for each compiler target machine. An Optimizer/Code Generator pair is referred to as COGN. COGN operates in a two-pass manner within a single phase. Optimization is performed during the first pass where the IL is translated to a modified IL. Code generation and register assignment are performed during the second pass, where the modified IL is transformed into the Code File.

The Editor (EDIT) reads the Code File and produces the relocatable object program file. EDIT also optionally generates an edited object code listing and a cross-reference listing. There is one EDIT phase per target computer.

The Compool Processor (COP) executes only in a compool build compilation. It executes after EDIT and transforms the symbol table contents into a Compool File for later inclusion in compilations which refer to compool-declared entities.

The major compiler data base elements include the Symbol Table, Compilation Control Block, Intermediate Language File and Code File. The Symbol Table consists of a fixed-length hash table and a variable-length set of entries that describe the structures and constructs that are derived from source language parsing (e.g., source-program-declared tables, blocks, and procedures) and that are produced to assist the compiling process (e.g., compiler-generated labels and items). There are two types of symbol table entries: name entries and attribute entries. Some attribute entries describe source program entities which have names, such as variables, procedures, and labels. Since names are variable in length, and since more than one entity can bear the same name, the name part of a symbol table entry is maintained separately from the related attribute information that describes the entity.

The Compilation Control Block (CCB) is a small core-resident block of data used for communication between the compiler executive and the phases. CCB is a collection of items such as symbol table chain headers, a bit vector describing compilation options, and the current statement number.

The Intermediate Language (IL) File represents the executable statements of a program in Polish postfix form. It is produced by the syntax analyzer phase, and is read by the optimizer/code generator phase. The IL was designed to simplify optimization and code generation. It has the following characteristics:

AD-A055 591 RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 1/3
AFAL SIMULATION FACILITY/CAPABILITY MANUAL. VOLUME I. EXECUTIVE--ETC(U)
JUN 77 R A WHISNANT, W H RUEDGER, R L EARP F33615-76-C-1308
UNCLASSIFIED AFAL-TR-77-118-VOL-1 NL

2 OF 5
AD
A055 591

1. The IL is, in general, language and machine independent;
2. All conversions are explicitly expressed in the IL;
3. The IL is, in general, nonredundant; for example, a FOR statement is expressed in terms of simpler statements such as assignment, IF, and GOTO.

The Code File (CF) is used for three purposes: (1) it contains name set/use information produced by ANZR for the cross-reference listing; (2) it contains variable preset values produced by ANZR; (3) it contains the target machine instructions produced by the Code Generator. The Code File is read by the Editor.

2.1.2.2.1.2 FORTRAN. The FORmula TRANslator language, FORTRAN, is a widely used procedure-oriented programming language. It is designed for solving scientific-type problems and is thus composed of mathematical-like statements constructed in accordance with precisely formulated rules. Therefore, programs written in the FORTRAN language consist of meaningful sequences of these statements that are intended to direct the computer to perform the specified computations. DEC-10 FORTRAN-10 is compatible with and encompasses an ANSI standard. FORTRAN-10 also provides many extensions and additions to this standard which greatly enhance its usefulness and increase its compatibility with other FORTRAN language sets. Extensions include subroutines which allow the FORTRAN user to do real-time programming. With these subroutines, the time-sharing job can dynamically connect real-time devices to the priority interrupt (PI) system, respond to these devices at interrupt level, remove the devices from the PI system, and change their PI level. Use of these routines requires that the user have real-time privileges and be able to lock his job in core.

FOROTS, the FORTRAN-10 object-time system, implements all program data file functions and provides the user with an extensive runtime error reporting system. An additional feature is that the association between FORTRAN logical units and the file descriptions to which they refer may be either made within the source program or deferred until runtime. DEC-10 FORTRAN-10 also supports FORDDT, an interactive program that is used as an aid in debugging FORTRAN programs.

2.1.2.2.1.3 ALGOL. The ALGOrithmic Language, ALGOL, is a scientific language designed for describing computational processes, or algorithms. It is a problem-solving language in which the problem is expressed as complete and precise statements of a procedure. The DEC-10 ALGOL system is based on ALGOL-60. It is composed of the ALGOL processor, or compiler, and the ALGOL object time system. Any errors made in writing the program are detected by the compiler and passed on to the user.

The ALGOL object time system provides special services, including the input/output service, for the compiled ALGOL program. Part of the object time system is the ALGOL library, a set of routines that the user's program can call in order to perform calculations. These include the mathematical functions and the string and data transmission routines. These routines are loaded with the user's program when required: the user need only make a call to them. The remainder of the object time system is responsible for the running of the program and providing services for system resources, such as core allocation and management and assignment of peripheral devices.

2.1.2.2.1.4 APL. A Programming Language (APL) is a concise programming language especially suitable for dealing with numeric and character array-structured data. APL is a completely conversational system which tends to increase programmer productivity and expertise by allowing the user to interact with the APL system and his running programs. APL is rich in operators that facilitate array calculations. This higher-level programming is accomplished by suppressing much of the programming detail inside single APL operators. One operator may be used to sort a vector of values in ascending order, thereby making "sort" a primitive operation rather than a tedious subroutine. APL is intended for use as a general data processing language as well as a mathematician's tool.

2.1.2.2.1.5 BASIC. The Beginner's All-purpose Symbolic Instruction Code, BASIC, is a problem-solving language that is easy to learn because of its conversational nature. It is particularly suited to a time-sharing environment because of the ease of interaction between the user and the computer. The BASIC language can be thought of as divided into two sections: one section of elementary statements that the user must know in order to write simple programs, and a second section of advanced techniques for more powerful programs.

The BASIC system has several special features built into its design:

1. BASIC contains its own editing facilities. Existing programs and data files can be modified directly with BASIC instead of with a system editor by adding or deleting lines, by resequencing the line numbers, or by combining two files into one. The user can request a listing of all or part of any of his files on either the line printer or the terminal.
2. At the editing level, BASIC allows various peripheral devices to be used for storage or retrieval or programs and data files; within a program, information can be read from or written to the terminal and to the disk (in the latter case, either sequentially or by random access);
3. Output to the terminal can be simply formatted by tabs, spaces, and column

headings or more precisely formatted by using the advanced PRINT USING statement;

4. BASIC has statements designed exclusively for matrix computations;
5. An advanced string handling capability includes a concatenation operator, substring and search functions, and other string intrinsic functions; mathematical intrinsic functions are contained in BASIC, along with methods by which the user can define his own functions.

2.1.2.2.1.6 AID. The Algebraic Interpretive Dialogue, AID, is the DEC-10 adaption of the language elements of JOSS, a program developed by the RAND Corporation. To write a program in the AID language requires no previous programming experience. Commands to AID are typed in via the user's terminal as imperative English sentences. Each command occupies one line and can be executed immediately or stored as part of a routine for later execution. The beginning of each command is a verb taken from the set of AID verbs. These verbs allow the user to read, store, and delete items in storage; halt the current processing and either resume or cancel execution; type information on his terminal; and define arithmetic formulas and functions for repetitive use that are not provided for in the language. However, many common algebraic and geometric functions are provided for the user's convenience.

The AID program is device-independent. The user can create external files for storage of subroutines and data for subsequent recall and use. These files may be stored on any retrievable storage media, but for accessibility and speed, most files are stored on disk.

2.1.2.2.1.7 COBOL. The COmmon Business Oriented Language, COBOL, is an industrywide data processing language that is designed for business applications, such as payroll, inventory control, and accounts receivable.

Because COBOL programs are written in terms that are familiar to the business user, he can easily describe the formats of his data and the actions to be performed on this data in simple English-like statements. Therefore, programming training is minimal. COBOL programs are self-documenting, and programming of desired applications is accomplished quickly and easily.

DEC-10 COBOL accepts two source program formats: conventional format and standard format. The conventional format is employed when the user desires his source programs to be compatible with other COBOL compilers. This is the format normally used when input is from the card reader. The standard format is provided for users who are

familiar with the format used in DEC-10 operations. It differs from conventional format in that sequence numbers and identification are not used because most DEC-10 programs require neither.

2.1.2.2.1.8 DBMS. The Data Base Management System (DBMS-10) is a facility of the DEC-10 that permits the user to consolidate his data files into one or more data bases. A data base is a collection of nonredundant data items that can be accessed by a variety of programs and/or applications that have common processing requirements and functional relationships. The data base is created and maintained through modules of DBMS-10. These modules permit the user to structure the data in such a way that each application can access in an optimum fashion, yet no data item is actually duplicated in the data base. This arrangement is accomplished by the data administrator who structures the data base in a manner such that each application can access it through a search pattern most suited to its needs. Once the data base has been established, users can access the data through COBOL programs containing special data base syntax.

2.1.2.2.2 Utility software

2.1.2.2.2.1 MACRO Assembler. MACRO is the symbolic assembly program on the DEC-10. It generates machine language programs by performing the following functions:

1. Translating symbolic operation codes in the source program into the binary codes needed in machine language instructions;
2. Relating symbols specified by the user to numeric values;
3. Assigning absolute core addresses to the symbolic addresses of program instructions and data;
4. Preparing an output listing of the program which includes any errors detected during the assembly process.

MACRO is a two-pass assembler. This means that the assembler reads the source program twice. Basically, on the first pass, all symbols are defined and placed in the symbol table with their numeric values, and on the second pass, the binary (machine) code is generated. Although not as fast as one-pass assembler, MACRO is more efficient in that less core is used in generating the machine language code and the output to the user is not as long.

MACRO is a device-independent program; it allows the user to select, at runtime, standard peripheral devices for input and output files. For example, input of the source program can come from the user's terminal and output of the assembled binary program can

go to a magnetic tape, and output of the program listing can go to the line printer. More commonly, the source program input and the binary output are disk files.

2.1.2.2.2.2 Linking Loader. LINK-10, the DEC-10 linking loader, merges independently translated modules of the user's program into a single module and links this module with system modules into a form that can be executed by the operating system. It provides automatic relocation and loading of the binary modules producing an executable version of the user's program. When the loading process has been completed, the user can request LINK-10 either to transfer control to his program for immediate execution or to output the program to a device for storage in order to avoid the loading procedure in the future.

While the primary output of LINK-10 is the executable version of the user's program, the user can request auxiliary output in the form of map, log, save, symbol, overlay plot, and expanded core image files. This additional output is not automatically generated; the user must include appropriate switches in his command strings to LINK-10 in order to obtain this type of output. The user can also gain precise control over the loading process by setting various loading parameters and by controlling the loading of symbols and modules. Furthermore, by setting switches in his command strings to LINK-10, the user can specify the core sizes and starting addresses of modules, the size of the symbol table, the segment into which the table is placed, the messages he will see on his terminal or in his log file, and the severity and verbosity levels of the messages. Finally, he can accept the LINK-10 defaults for items in a file specification or he can set his own defaults that will be used automatically when he omits an item from his command string. LINK-10 has an overlay facility to be used when the total core required by a user's program is more than the core available to the user.

2.1.2.2.2.3 Program Debugging. The Dynamic Debugging Technique, DDT, is used for on-line program composition of object programs and for on-line checkout and testing of these programs. For example, the user can perform rapid checkout of a new program by making a change resulting from an error detected by DDT and then immediately executing that section of the program for testing.

After the source program has been compiled or assembled, the binary object with its table of defined symbols is loaded with DDT. In command strings to DDT, the user can specify locations in his program, or breakpoints, where DDT is to suspend execution in order to accept further commands. In this way, the user can check out his program section-by-section and if an error occurs, the user can insert the corrected code immediately.

2.1.2.2.2.4 File Manipulation. The Peripheral Interchange Program, PIP, is used to transfer data files from one I/O device to another. Commands to PIP are formatted to accept any number of input (source) devices and one output (destination) device. Files can be transferred from one or more source devices to the destination device as either one combined file or individual files. Switches contained in the command string to PIP provide the user with the following capabilities:

1. Naming the files to be transferred,
2. Editing data in any of the input files,
3. Defining the mode of transfer,
4. Manipulating the directory of a device if it has a directory,
5. Controlling magnetic tape and card punch functions,
6. Recovering from errors during processing.

2.1.2.2.2.5 File Editing. The Text Editor and COrrector program, TECO, is a powerful editor used to edit any ASC11 text file with a minimum of effort. TECO commands can be separated into two groups: one group of elementary commands that can be applied to most editing tasks, and the larger set of sophisticated commands for character string searching, text block movement, conditional command, programmed editing, and command repetition.

TECO is a character-oriented editor. This means that one or more characters in a line can be changed without retyping the remainder of the line. TECO has the capability to edit any source document: programs written in MACRO, FORTRAN, COBOL, ALGOL, or any other source language; specifications; memorandums, and other types of arbitrarily formatted text. The TECO program does not require that line numbers or other special formatting be associated with the text. Editing is performed by TECO via an editing buffer, which is a section within TECO's core area. Editing is accomplished by reading text from any device into the editing buffer (inputting), by modifying the text in the buffer with data received from either the user's terminal or some other device (inserting), and by writing the modified test in the buffer to an output file (outputting).

2.1.2.2.2.6 Manuscript Editing. RUNOFF facilitates the preparation of typed or printed manuscripts by performing line justification, page numbering, titling, indexing, formatting, and case shifting as directed by the user. The user creates a file with an editor and enters his material through his terminal. In addition to entering the text, the user includes information for formatting and case shifting. RUNOFF processes the file and produces the final formatted file to be output to the terminal, the line printer, or to another file.

With RUNOFF, large amounts of material can be inserted into or deleted from the file without retyping the text that will remain unchanged. After the group of modifications have been added to the file, RUNOFF produces a new copy of the file which is properly paged and formatted.

2.1.3 Special Purpose Peripherals

In order to provide the capability for simulations which assess the influence of human factors on avionic system performance, the AVSAIL laboratory includes a simulated cockpit and computer-generated visual displays of the external environment and the cockpit flight data displays. The functional configuration of these elements of AVSAIL is shown in Figure 2.1.3-1. The elements shown are configured in three basic subsystems, which are referred to here as (1) the Picture System, (2) the Video System, and (3) the Cockpit Simulator. Overall communication between these special purpose peripherals is through the DEC-10 which executes the simulation models. Capabilities of each of these three peripheral subsystems are described in succeeding sections.

2.1.3.1 The Picture System

The Picture System is a standalone general purpose, interactive computer graphics system which can display smoothly moving pictures of two- or three-dimensional objects effectively in real time. The basic components of the system, manufactured by the Evans & Sutherland Computer Corporation, are a DEC PDP-11; hardware processing units which perform such functions as rotations, zooming, and perspective; an 8172-point Refresh Buffer; a Picture Generator; a Character Generator; a 21 in. Picture Display; a Tablet to facilitate picture interaction; and the software to support the system.

Figure 2.1.3.1-1 is an overall view of the Picture System interfaced with a DEC PDP-11/50 computer. A close-up view of the Input Tablet, and 21 in. Picture Display (with a typical display configuration) is shown in Figure 2.1.3.1-2. The tablet serves as the standard, general-purpose, graphic input device in THE PICTURE SYSTEM. The tablet can be used for positioning or pointing to the picture elements by use of a pen whose x and y coordinates are read by the picture controller. In this manner, the tablet and pen can be used to simulate functions, such as joy stick control, such that the operator can interactively "fly the simulation." An operator seated at the Input Tablet is shown "flying the simulation" in Figure 2.1.3.1-3.

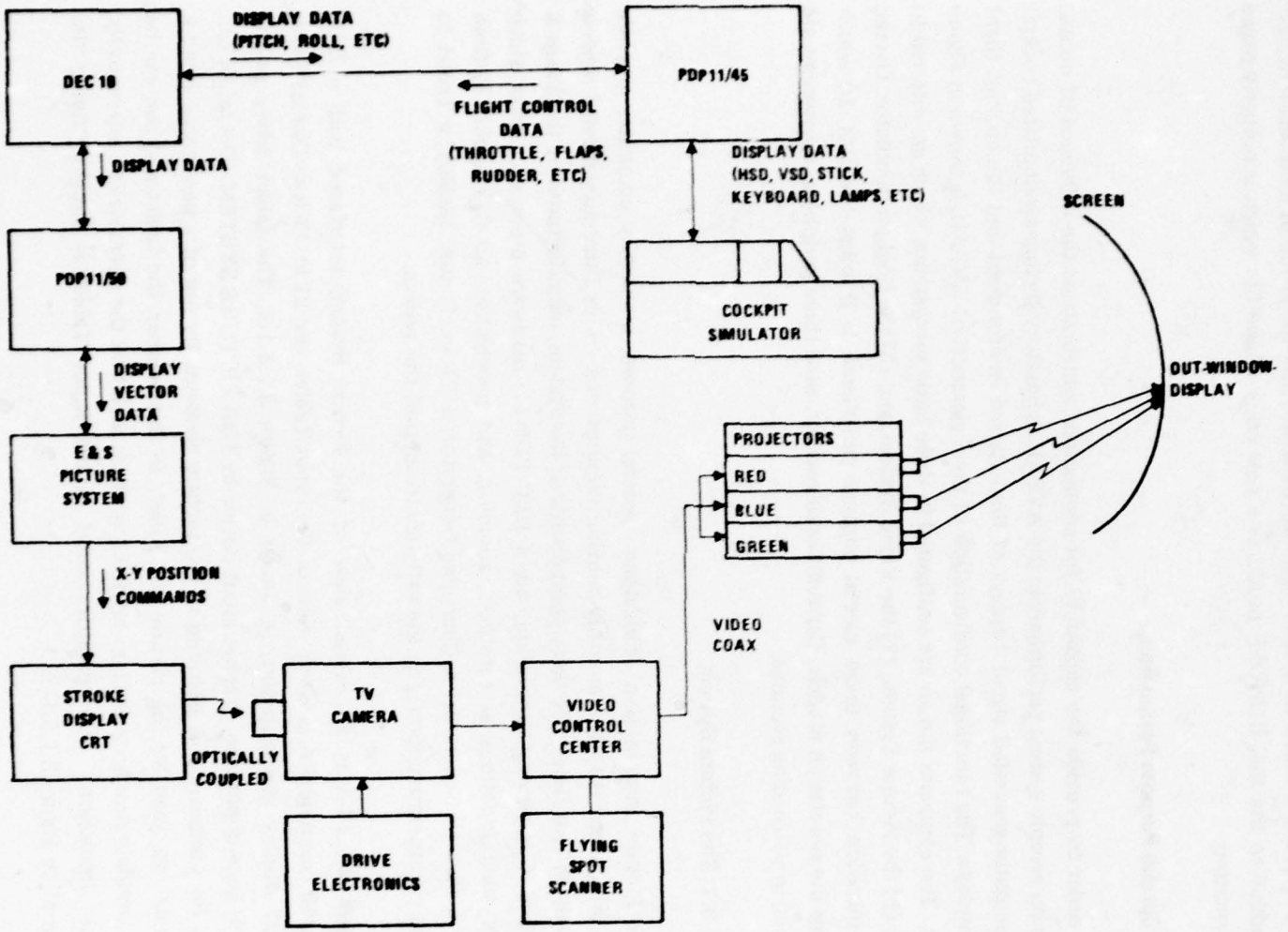


Figure 2.1.3-1. Special purpose peripherals functional configuration.

Figure 2.1.3.1-1. The Picture System.

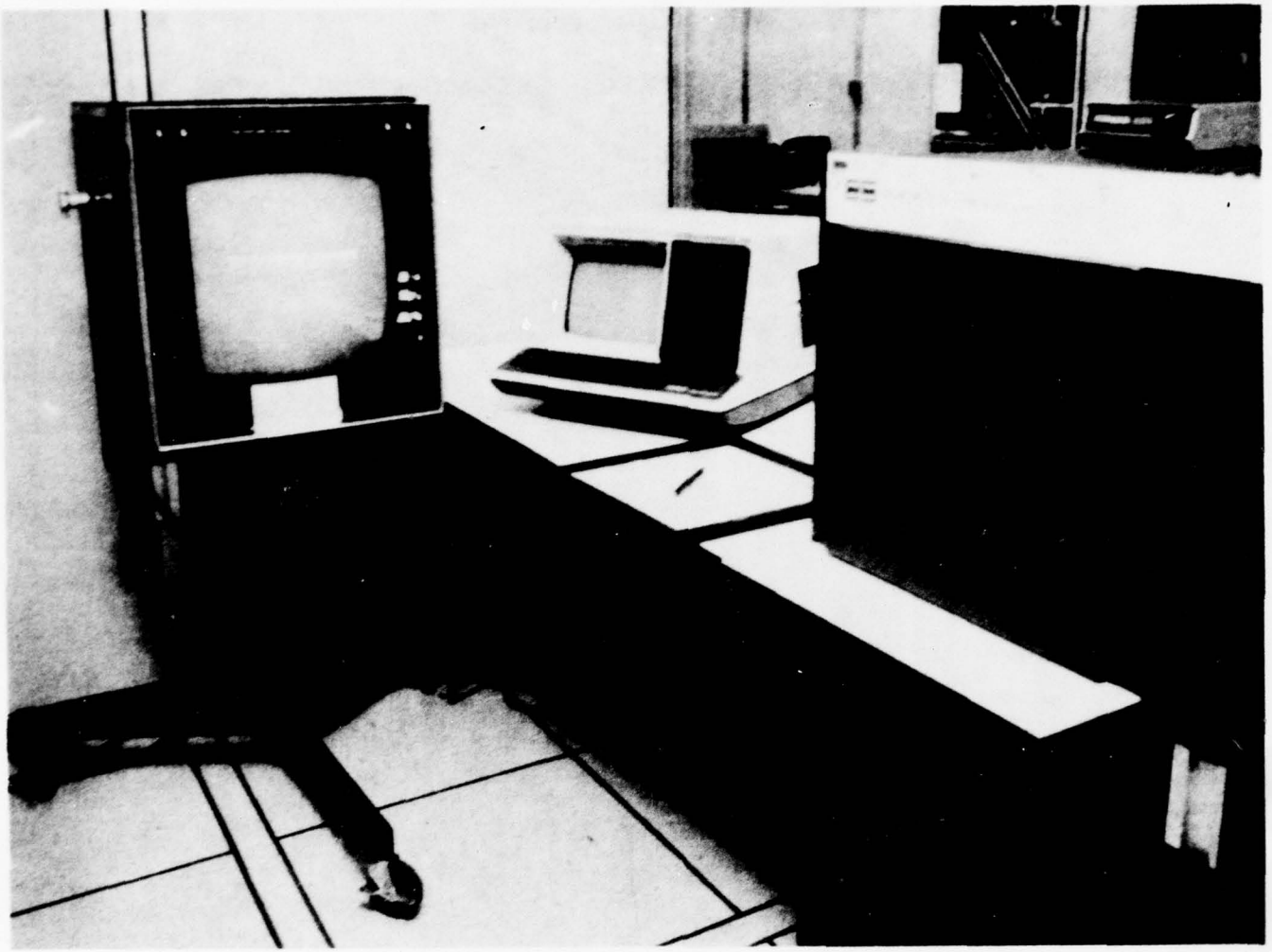


Figure 2.1.3.1-2. The Picture display and tablet.

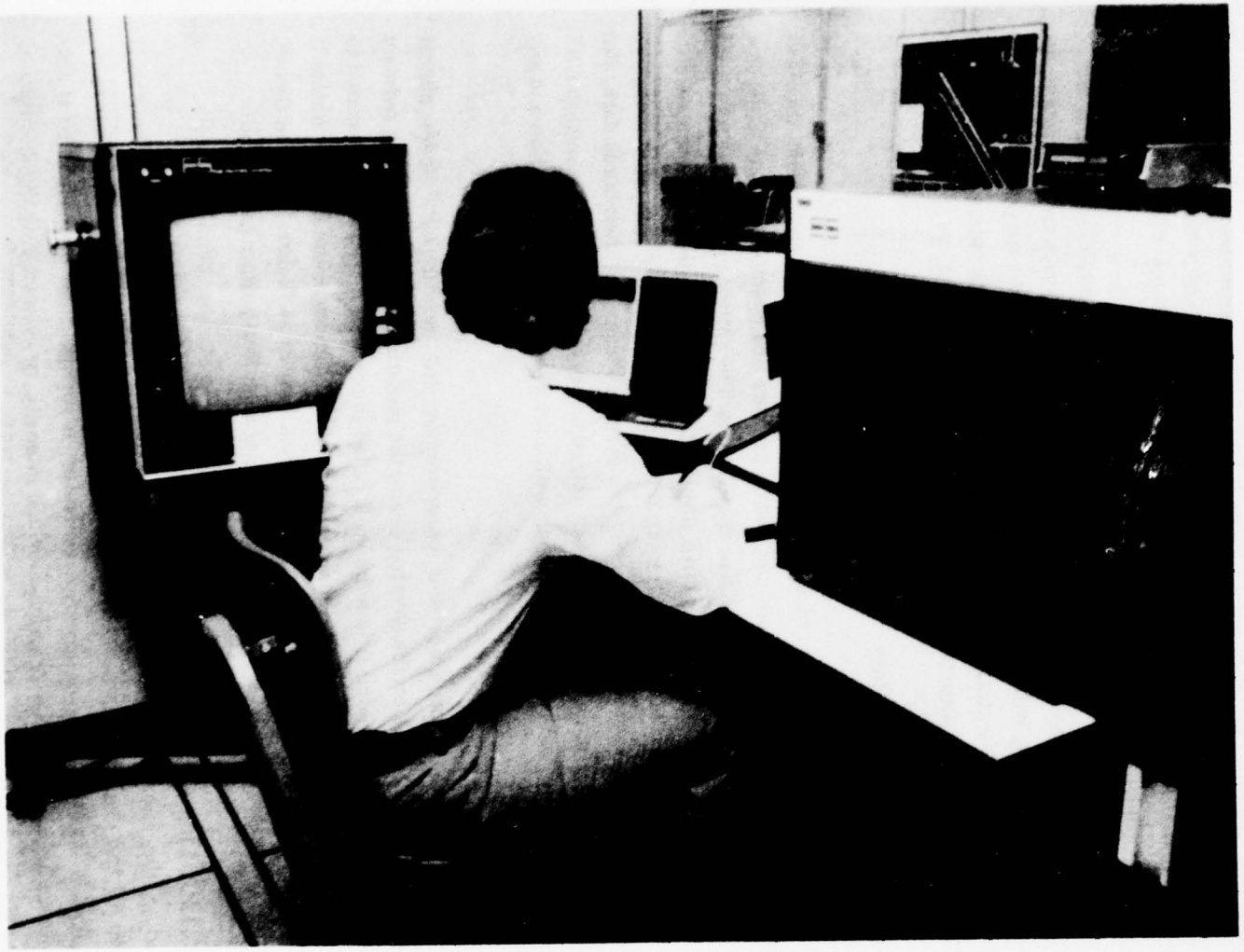


Figure 2.1.3-1-3. An operator flying the simulation.

2.1.3.1.1 Overview of interactive computer graphics

Interactive computer graphics allows a user to dictate changes to the picture and see the results immediately. The system time lag is a very small fraction of a second, and the user gets the feeling that he is actually manipulating the picture itself in real time.

Computer graphics is a very broad subject, encompassing many details which are not pertinent here. However, some appreciation of the more basic aspects, as represented by the AVSAIL Picture System, will help to orient the reader. Four topic areas are presented in subsequent sections: presenting a prepare picture, representing structures to be depicted, preparing a picture of such structures, and interacting with the picture.

2.1.3.1.2 Picture presentation

2.1.3.1.2.1 Graphical Output Media. Output media fall into two basic divisions, permanent and impermanent. Plotters and roster printers are examples of the first type, which do not lend themselves to interactive applications. The cathode ray tube (CRT) is the most widely used impermanent, interactive display device. Information is presented on a CRT by directing a beam of electrons about on its phosphor coated face. The CRT face emits light for an instant when it is struck by the electron beam and then turns dark. For the picture to be visible it must be redrawn or refreshed very frequently. The refresh CRT used by the Picture System can be drawn upon with a set of strokes at any position and any angle.

2.1.3.1.2.2 Refresh Rate. Since the phosphor on the refresh CRT fades almost immediately after it is struck by the electron beam, the picture must be continually redrawn to be viewed. This rate at which it is redrawn is called the refresh rate, usually measured in frames per second. If the picture is not redrawn frequently enough, the eye will notice it fading between refreshes, producing an unsightly effect known as flicker. To avoid flicker, the Picture System is refreshed at a rate which is greater than thirty times per second.

2.1.3.1.2.3 *Line* Generation. A line is specified by two end-points (x,y) and (x',y'), expressed in the coordinate system of the CRT, called screen coordinates. The actual movement of the electron beam between the two points is accomplished by a hardware device called a line generator or a vector generator. A sophisticated line generator is also capable of drawing lines with a program-specified intensity, or even varying the intensity of a line from one end to the other. In this most general case, where line endpoints are specified by the three coordinates (x,y,z), the intensity or brightness of lines can appear to

trail off in the distance, producing an illusion of depth. This technique is known as depth-cueing.

2.1.3.1.2.4 Update Rate. The advantage of the refresh CRT is that it can show smoothly changing pictures. Lines drawn on a CRT do not really move, of course, but the illusion of motion is imparted by continually redrawing the picture of each frame with lines at slightly different positions each time. The eye blends this sequence of slightly different frames together into a smoothly moving picture such as a motion picture. The rate at which these different frames can be displayed is called the update rate. In contrast to the refresh rate which counts the number of pictures drawn per second, whether or not they are changed, the update rate counts only those frames that are different. An update rate of 10-20 frames per second will provide smooth motion.

2.1.3.1.2.5 Picture Buffering. In the Picture System a refresh buffer provides storage so that the refresh and update rates may be different. Although refresh of 30-40 frames per second is required to avoid flicker, update of 10-20 frames per second is adequate tc provide smooth motion. In effect, each new frame is shown two, three, or even four times while the next frame is being computed.

Data resident in a refresh buffer is called a Display File. Full frames stored in this buffer may be read out and used to refresh the CRT any number of times before a new frame is created. Typically, new frames are created 20 times a second and the picture is refreshed 40 times a second; i.e., each frame is shown twice. Thus, the presence of a refresh buffer allows both refresh and update to proceed at their respective optimal rates and the system has a larger line capacity than it otherwise would.

A potential problem area exists when a picture is refreshed from a memory which is simultaneously being filled with a new frame, namely, that a picture displayed may consist of some lines from one frame and some from another. This can produce a number of effects, some very unsightly. To avoid this problem, the refresh buffer can be split into two separate buffers, and update and refresh can be switched between the two in a way which avoids conflicts. This is called double-buffering, and its only disadvantage is that the amount of pictorial data which may be buffered is halved. In some cases this can place an unnecessarily low ceiling on the line capacity. The alternative, single buffering, can be used to take advantage of the entire buffering space when the effects are not too disturbing, usually when the pictures shown are not highly dynamic.

2.1.3.1.3 Picture definition

Data ultimately deposited in a refresh buffer must originate in the memory of the computer controlling the system. This computer-resident data is called a Data Base and may be vastly different in form from the display file which emanates from it.

The data base contains the coordinates of points in the structure to be displayed, along with instructions for interpreting those points. Along with coordinate information there may be pointers, substructure names, and other non-graphic information and attributes.

Points are the basic geometric entities in the data base. There are three basic instructions for treating a point: move the beam to that point, draw a line to that point, or draw a dot at that point.

The most straightforward way to specify the position of a point is simply to state its absolute coordinates. An alternative that often introduces considerable efficiencies, called relative coordinates, entails stating the displacement required to get to a point from the previous point. Codes for common sequences like "absolute, relative, absolute, relative. . . ." can be made recognizable to facilitate handling tables of points.

If a structure to be displayed lies in a plane, it is simplest and most efficient to define it using two-dimensional data. In this case it is typical to supply an x and a y coordinate for each point in the structure, and then perhaps a single z coordinate which applies to all the points.

If however, the structure is non-planar, it must be defined as three-dimensional data where a coordinate triple of the form (x,y,z) is given for each point.

In general a full computer word is devoted to each coordinate of each point and all coordinates are expressed as integers. In the 16-bit computer, then, the largest expressible positive number is 32767. This is sufficient for many applications, but the need to express larger numbers sometimes arises. This need can be met, at the expense of some loss of resolution in data definition, by employing an alternate means of expressing data called homogeneous coordinates. Here a point (x,y,z) is defined by the four coordinates (hx,hy,hz,h· 32767), where h is an arbitrary number between zero and one. It is apparent though that resolution is lost; when h is 1/2, it is impossible to exactly express odd values for the original coordinates. Smaller values of h impose a correspondingly greater loss of resolution.

It is customary to conserve core by supplying only the first three coordinates (hx,hy,hz) for three-dimensional points, or just two coordinates (hx,hy) for two-dimensional points (with a common value for hz), and to prespecify a fourth coordinate (usually referred to as w) which applies to several such points.

2.1.3.1.4 Picture preparation

The data base is almost never identical to the display file because the base represents some view of that scene. To create a display file, transformation of the data base is required. In order to prepare a structure for display, it may have to be changed in size, position, or orientation; it may have to be put in perspective as seen from a given vantage point; parts of it may have to be removed to keep everything within a given field of view; and its coordinate system may have to be changed to conform with the output device. All of these steps can be expressed mathematically and implemented in software or hardware.

Fortunately, since many of the steps involved in picture preparation are invariant from application to application, it is very worthwhile to implement them with special purpose hardware. Any calculations unique to a given application can still be performed in software. To meet the demand for fast frame creation, high performance graphic systems employ special purpose hardware processors to implement the picture preparation steps. These steps are described in the following sections.

2.1.3.1.4.1 Simple Linear Transformations. Linear transformations (rotations, translations, scalings, etc.) can be described by parameters which indicate the type and degree of information. If the transformation parameters are properly arranged into a matrix, a vector of original coordinates can be multiplied by this matrix to yield a vector of new coordinates reflecting the desired transformation.

A 4 x 4 matrix can represent any rotation, translation, or change in scale and can be used to transform points represented by homogeneous coordinates or, as special cases, two-dimensional or three-dimensional coordinates.

2.1.3.1.4.2 Compound Linear Transformations. All linear transformations can be expressed as a sequence of simple translations, rotations, and changes in scale. A transformation expressible only by such a sequence is called a compound transformation. When a compound transformation is to be applied to a set of points, first a composite matrix is formed by multiplying together matrices representing all the simple transformations in the sequence, in the same order in which the data would have

encountered the original transformations, and then applying this composite matrix to all points to be transformed. The process is known as transformation concatenation.

2.1.3.1.4.3 Perspective. The perspective operation entails computing a point projection of three-dimensional points onto a plane representative of the screen, as depicted in Figure 2.1.3.1-4. Perspective can be applied to three-dimensional data by taking advantage of the fact that the perspective transformation is expressible in matrix form: a perspective transformation matrix can be included at the end of the sequence of rotation, translation, and scale matrices to transform three-dimensional data into a two-dimensional perspective representation.

2.1.3.1.4.4 Windowing. In some graphics applications, the data base is displayed in its entirety on the screen. Often, however, a closeup of some portion of the data base is desired and the rest is preferably omitted. Determining what to omit is so time-consuming in software that it jeopardizes the dynamic movement of the picture.

The Picture System can address this so-called windowing problem by performing a *visibility check* in hardware after the transformation stage and drawing only visible lines on

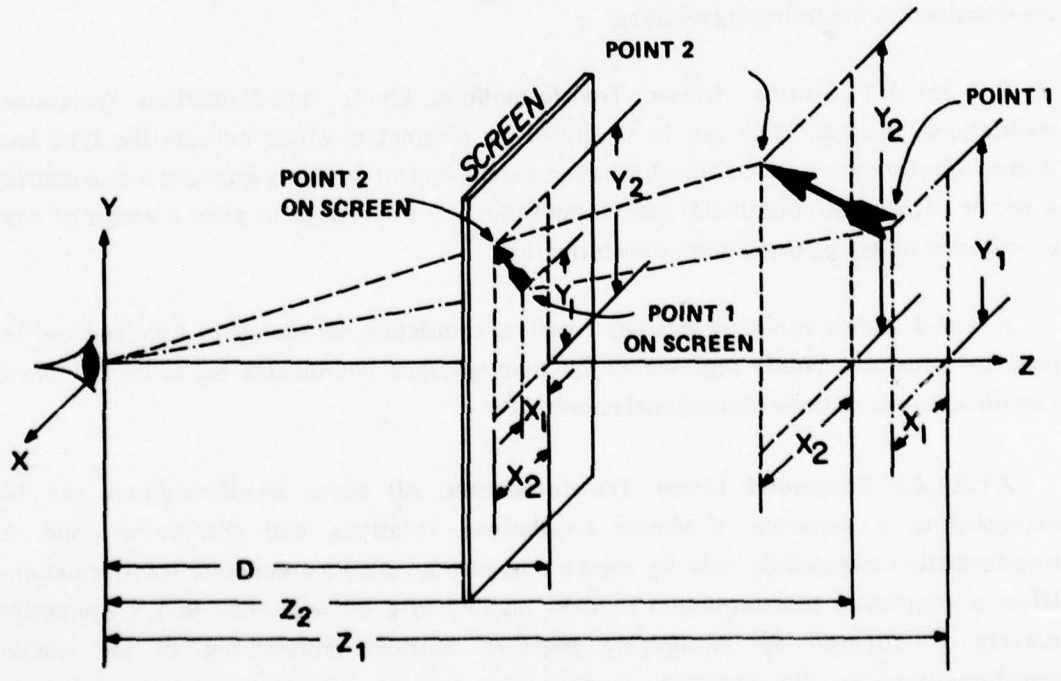


Figure 2.1.3.1-4. Three-dimensional perspective projections onto a two-dimensional plane.

the display. One implementation of windowing is called clipping, and entails comparing all lines with the boundaries of a program-specified field of view superimposed on the data base. Lines or portions of lines outside the field of view are eliminated and only visible lines are passed on for display on the screen.

In two dimensions, *the field of view is a rectangle called a window*, superimposed on the plane of the data base. Clipping is easiest if the sides of the rectangle are parallel with the coordinate axes; however, this presents no restriction since the effect of a rotated window can be obtained by rotating the data in the opposite direction.

A window is specified by supplying values for its left, right, bottom, and top boundaries using the same coordinate system used in the data base. Two-dimensional clipping is diagrammed in Figure 2.1.3.1-5.

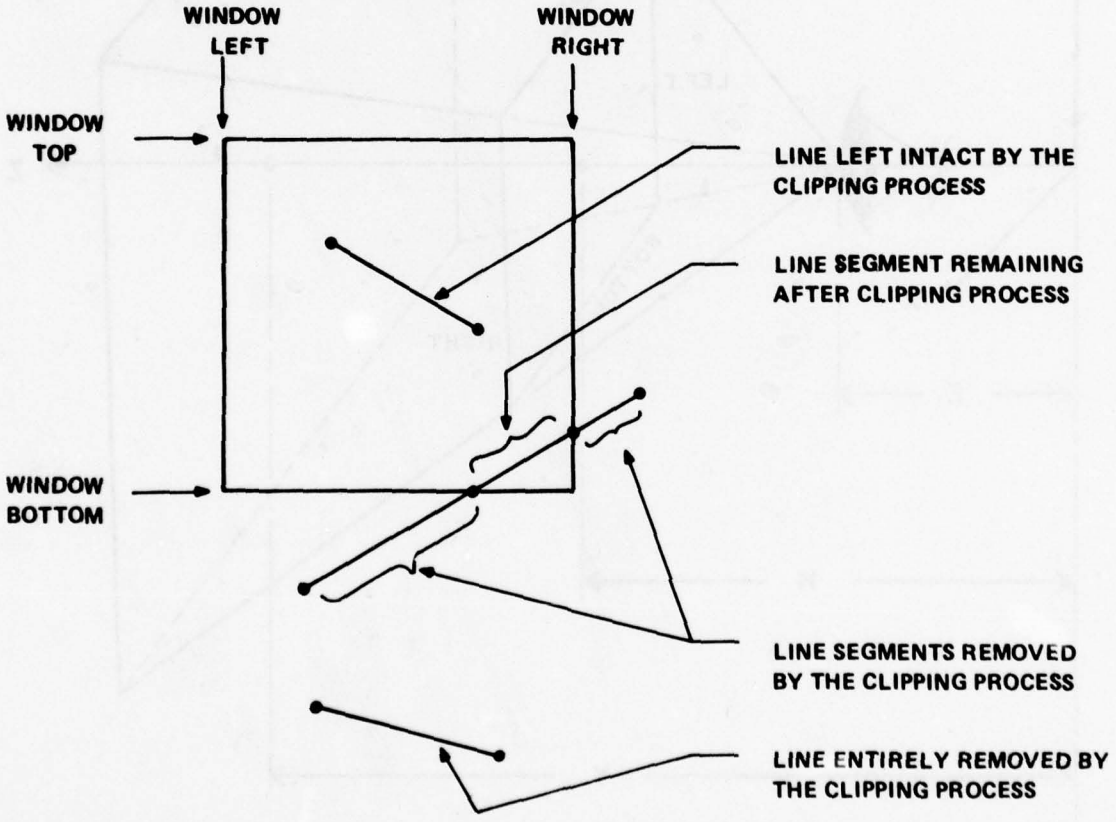


Figure 2.1.3.1-5. Two dimensional clipping.

In three dimensions the field of view is a three-dimensional region. It may be a rectangular volume, or, if its contents are to be seen in perspective, a section of a pyramid called a frustum of vision. Such a frustum is shown in Figure 2.1.3.1-6 along with the parameters necessary to completely specify it.

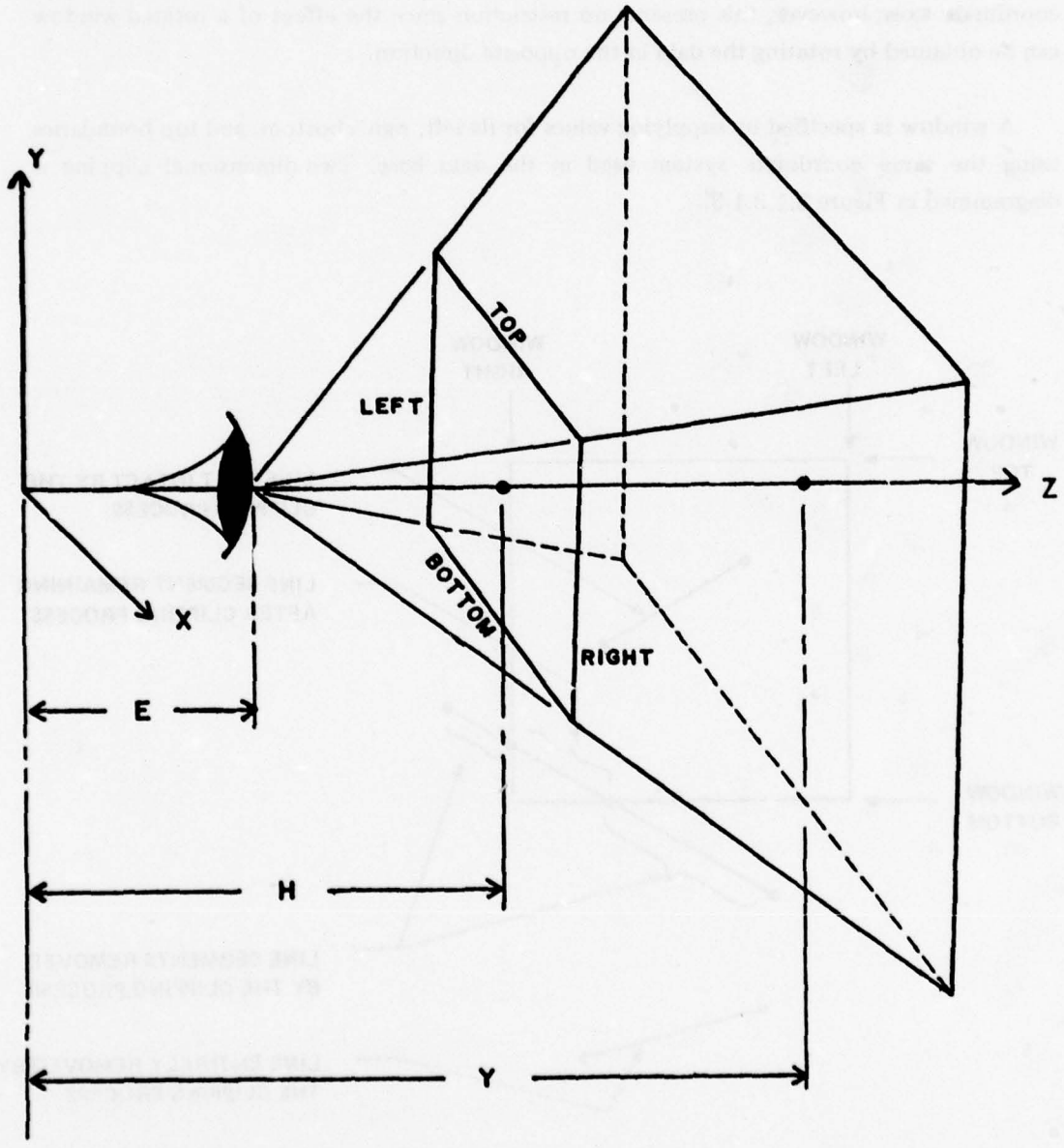


Figure 2.1.3.1-6. Frustum of vision showing the eye position in relation to an arbitrary coordinate axis.

In Figure 2.1.3.1-6 an eye positioned at point E along the Z axis is to see the portion of the data base that lies within the frustum whose hither (near) boundary is at point H, yon (far) boundary is at point Y, and whose side boundaries are determined, as in the two-dimensional case, by the window left, right, bottom, and top boundaries at the hither plane.

As in the two-dimensional case, lines are retained, completely eliminated, or partially eliminated, depending on whether they are completely within, completely outside, or partially outside the frustum of vision.

Another approach to windowing is called scissoring. Scissoring entails making available a screen coordinate drawing space which is somewhat larger than the screen itself and then intensifying only the lines and line segments actually on the screen. Scissoring is easier to implement than clipping and does not take up time in the picture preparation stage. On the other hand, scissoring permits an effective drawing area only slightly larger than the screen as opposed to the vastly larger effective drawing area permitted by clipping. Another disadvantage of scissoring is that the line generator spends time tracing out all lines, both visible and invisible, which makes flicker occur more readily.

2.1.3.1.4.5 Conversion to Screen Coordinates. Coordinate data that is not rejected by the clipping process is within limits determined by the field of view which may be of any size and at any position in the data base definition space. However, it is generally undesirable to display that data in a corresponding size and position on the screen. Rather, the data should be properly scaled (or mapped) so that it fills some program-specified region on the screen called a viewport. This can be accomplished by performing a final processing step which linearly maps all data from the window to the viewport.

If the viewport is a rectangular region aligned with the screen axes, it can be specified by supplying the screen coordinates for its left, right, bottom, and top edges. If the system's line generator can draw lines of varying intensity, a viewport may also specify the intensity limits for the data displayed. These limits specify the intensities of the data at the hither and yon boundaries and are called the hither and yon intensities. When the hither and yon intensities are different, the intensity of the displayed picture elements varies between these limits, allowing an illusion of depth to be imparted to the picture. A viewport is used to specify the region of screen and the intensity limits for the data to which, in the most general case, the frustum of vision is mapped. Figures 2.1.3.1-7 and 8 show how data may be displayed within a viewport which is the entire screen or only a portion of it. Viewports may also be utilized to map data into the coordinates of devices other than a display. For example, viewport boundaries could be specified in the coordinate system of a plotter or

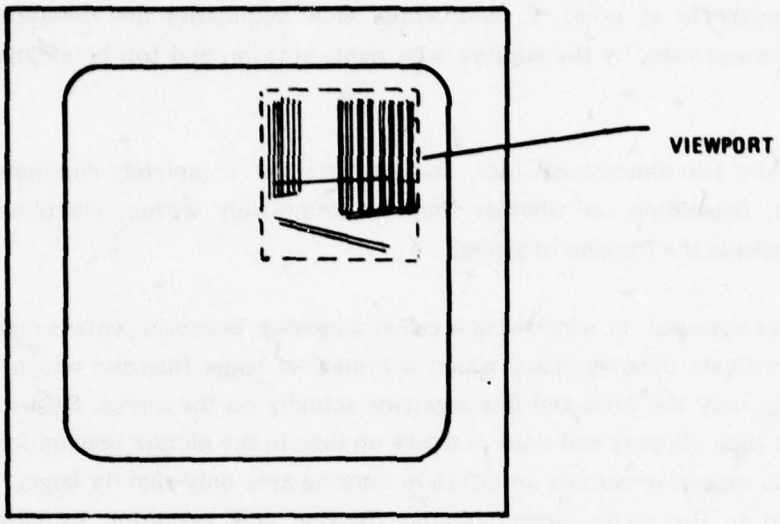


Figure 2.1.3.1-7. Partial screen viewport.

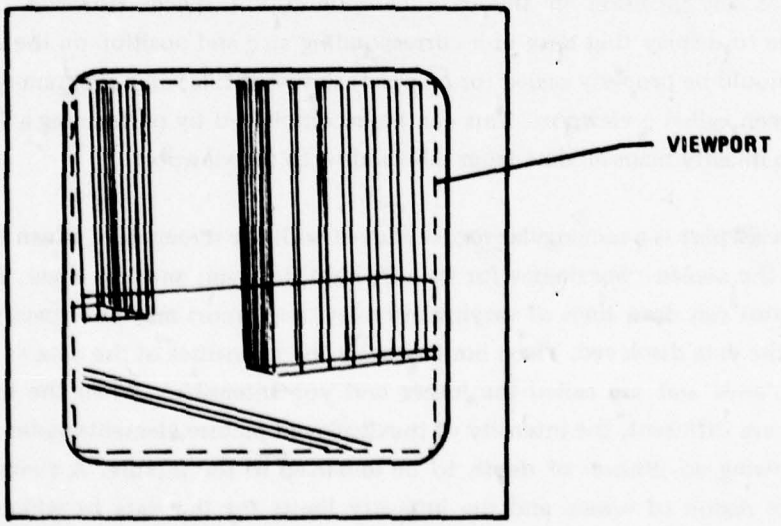


Figure 2.1.3.1-8. Full screen viewport.

similar device to provide the capability of obtaining hard copy output to the precision of the plotting device.

An advantage of program-specified viewports is that several may be assigned in the same program, each receiving different data. This technique proves convenient for many purposes in graphics, such as showing different views of an object or views in different directions from the same point on the same output device simultaneously.

2.1.3.1.4.6 Text Display. Almost all graphics applications call for the presentation of alphanumerics on the screen at one time or another. It is possible of course to define character shapes in the data base like other picture elements, and in fact, this is necessary if characters are to be treated like other objects, i.e., rotated, clipped, etc. However, it is possible to derive efficiencies from the foreknowledge of character properties when they do not require such sophisticated treatment, by generating the actual strokes of the characters just prior to drawing them and dealing only with character codes up to that point.

A hardware device which accepts character codes and produces the strokes comprising the character is called a Character Generator.

To use the generator to draw a string of characters, a display program must first stipulate character size, shape and orientation values; then position to where the string is to begin and insert a set of packed character codes, called a text string, into the display file. The Character Generator would then interpret the text string, look up the set of strokes associated with each code, size and orient the strokes properly, and draw the characters on the output device. Codes are packed into text strings as a memory conservation measure.

2.1.3.1.5 Picture interaction

Graphics applications require that the form or content of the picture be changeable by the user. A number of input devices for this purpose have been made available.

Function switches and lights are attached to the computer in the graphics system. These are toggle switches or push buttons from which polarity can be read. Each switch can be assigned a meaning unique to the program.

Analog input devices, including control dials, are also used for interaction. These devices offer one or more degrees of freedom over which a user can enter input values used for translation, scaling, etc.

A versatile interactive input device is the Tablet and Pen, which is a flat rectangular plate which may be positioned on a table in front of, or near, the display screen. Associated with the tablet is a pen which may be moved about over the plate. Its position on the plate may be read with fine resolution by the computer controlling the system. The computer can also detect whether the pen is actually touching the plate and may also indicate if the pen is near the plate. To tie pen motion together with a picture, a cursor is usually drawn on the screen. This cursor is a small symbol which continually moves about in concurrence with the pen. It soon becomes natural to guide the cursor to a desired position on the screen by an appropriate motion of the pen.

The tablet can also be programmed to perform the functions of function switches or the analog devices. In order to enable a tablet to perform the pointing function of typical light pen, the system should be equipped with a hit test feature which checks all data as it emerges from the transformation stage for proximity to the pen position. The user positions his cursor over the target structure and initiates the hit test feature (perhaps by touching the pen down). If a target structure is encountered, a flag is set which may be later tested or may be programmed to cause an interrupt. This method of pointing has the advantage that the target structure is marked in the data base, not the display file. It is often difficult or impossible to backtrack from an entry in the display file to find its corresponding entry in the data base.

The user of the tablet is allowed to sit in a natural writing position and at any desired distance from the graphic display. This reduces user fatigue and improves operating conditions.

2.1.3.1.6 Overview of the Picture System hardware

This section provides an overview of the hardware components which comprise the Picture System. A functional diagram of the configuration of the system is shown in Figure 2.1.3.1-9. The user of the system will normally interface with these components by means of the Graphics Software Package described in a later section.

2.1.3.1.6.1 The Picture Controller. The Picture Controller in the Picture System is a Digital Equipment Corporation PDP-11 General Purpose Digital Computer. Software available for the system includes a Text Editor, Macro Assembler, Linker, File Utility Packages, Debugging Packages, and higher level languages including BASIC and FORTRAN. The availability of these software systems and the Graphics Software Package provided with the Picture System enable the PDP-11 to act as the Picture Controller.

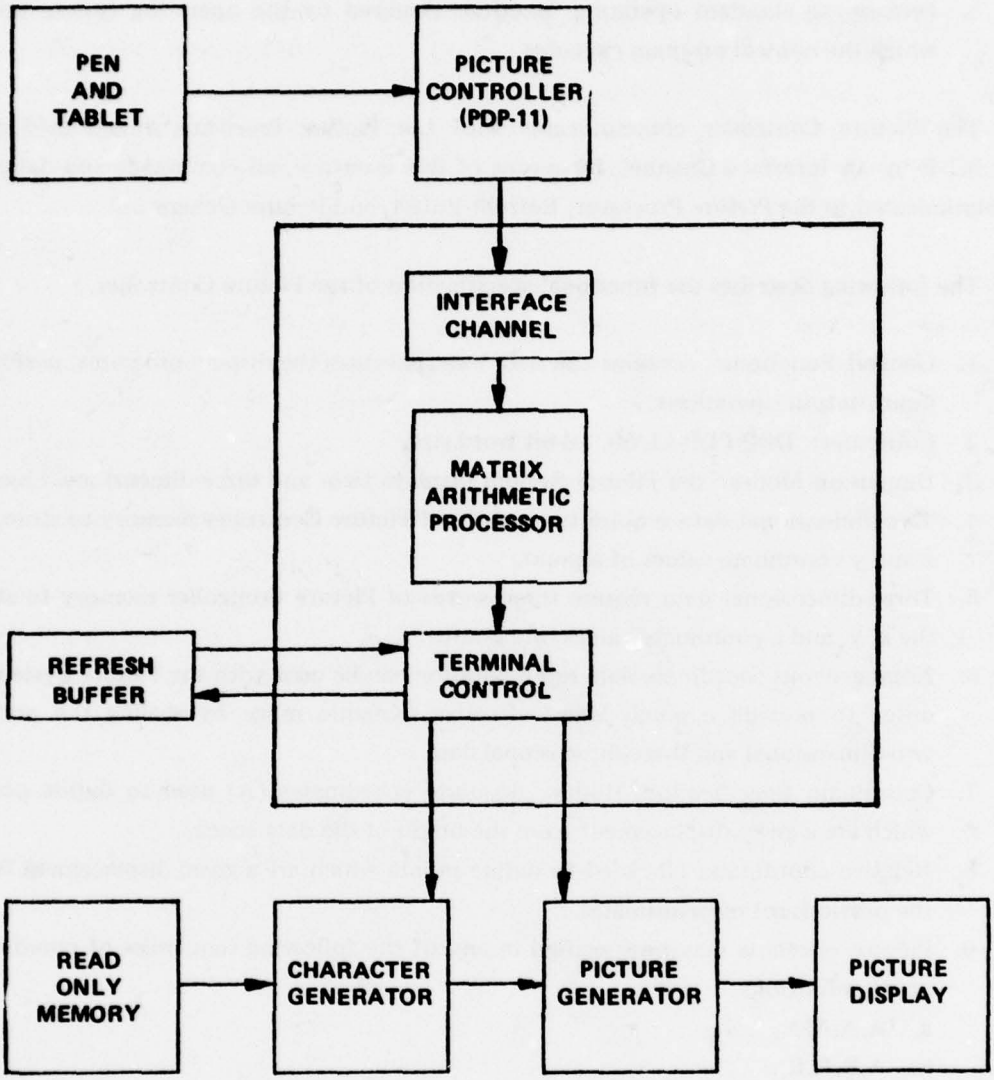


Figure 2.1.3.1-9. Functional configuration of Picture System.

The Picture Controller is used to:

1. Contain the data base which describes the object(s) to be viewed,
2. Control the processing of the object coordinate data by the Picture Processor,
3. Perform all input and output required to facilitate graphical interaction,
4. Compute parameters for use in simulation of object movement, data representation, etc.,

5. Perform all standard operating functions required by the operating system under which the control program executes.

The Picture Controller communicates with the Picture Processor shown in Figure 2.1.3.1-9 by an Interface Channel. By means of this interface, all commands and data are communicated to the Picture Processor, Refresh Buffer, and Picture Generator.

The following describes the functional specification of the Picture Controller.

1. General Functions: contains the data base, executes the display programs, performs input/output operations,
2. Computer: DEC PDP-11/50, 16-bit word size,
3. Dimension Modes: the Picture System displays two- and three-dimensional objects,
4. Two-dimensional data require two words of Picture Controller memory to store the x and y coordinate values of a point,
5. Three-dimensional data require three words of Picture Controller memory to store the x, y, and z coordinate values of a point.
6. Homogeneous coordinate data representation can be used with the Picture System in order to provide a much larger effective dynamic range by scaling the normal two-dimensional and three-dimensional data.
7. Coordinate Specification Modes: *Absolute* coordinates (A) used to define points which are a given displacement from the origin of the data space.
8. Relative coordinates (R) used to define points which are a given displacement from the previous set of coordinates.
9. Picture elements may be specified in any of the following sequences of coordinate point definitions:
   a. A,A,A,A,. . . ,
   b. A,R,R,R,. . . ,
   c. R,R,R,R,. . . ,
10. Drawing Modes: The *Move* (M) moves the beam position to a specified location with the beam intensity off.
11. The *Draw To* mode (DT) draws a straight line from the current beam position to a new specified location and leaves the beam position at a new location.
12. The *Dot* mode (D) moves the beam position to a specified location with the beam intensity off and then intensifies the beam at the specified location. The beam position remains at the dot location.
13. The *Character* mode (C) draws the specified character beginning at the current character beam position and then moves the beam position with intensity off to the position where the next character in a string begins.

14. Picture elements may be drawn using any of the above modes one by one, or they may be drawn using any of the following sequences of the above modes:
    a. M,DT,M,DT, . . . (unconnected lines),
    b. M,DT,DT,DT, . . . (lines connected end-to-end),
    c. DT,M,DT,M, . . . (another mode sequence for unconnected lines),
    d. DT,DT,DT,DT, ... (another mode sequence for lines connected end-to-end),
    e. D,D,D,D, . . . (a series of dots),
    f. C,C,C,C, . . . (a string of characters).
15. Instancing: A method of defining in the data base a two-dimensional or three-dimensional structure once and replicating it several times in a picture in different positions, sizes and orientations.
16. Instancing may be performed to any level.
17. Parameter Load/Store: The Picture Controller can load and store all control registers, status registers, and matrix registers that reside in the other components of The Picture System.

2.1.3.1.6.2 Matrix Arithmetic Processor. The Matrix Arithmetic Processor consists of a Transformation Matrix, a Transformation Matrix Stack, an Arithmetic Unit, and a Parameter Register File.

The Transformation Matrix is a 16-bit word. This 4 x 4 matrix is used to transform object coordinate data. It can also be concatenated with other 4 x 4 matrices to obtain a combined transformation.

The Transformation Matrix Stack is a storage area where up to four, 4 x 4 element matrices may be "stacked" or saved for future recall.

The Arithmetic Unit performs all arithmetic operations in the Picture Processor. This includes subtraction, addition, multiplication, division, and normalization.

The Picture Processor contains an array of 16-bit registers into which parameters specifying viewport boundaries, scale factors, etc., are stored and may be retrieved.

The Picture Processor utilizes these units to perform digital operations on the data received from the Picture Controller.

These operations are:

1. To process two-dimensional data;

2. To process three-dimensional data;
3. To push the Transformation Matrix onto the Matrix Stack;
4. To transfer the top 4 x 4 matrix of the Matrix Stack into the Transformation Matrix;
5. To load the Transformation Matrix with data from the Picture Controller's memory;
6. To store the contents of the Transformation Matrix into the Picture Controller's memory;
7. To concatenate the contents of the Transformation Matrix with a 4 x 4 matrix in the Picture Controller's memory to obtain a compound transformation;
8. To load and store the registers of the Picture Processor;
9. To check transformed coordinate data for visibility by comparison with a two-dimensional or three-dimensional viewing window—lines or portions of lines outside the window are removed by a clipping process so that only visible segments are processed further, and at this point three-dimensional data are converted to two dimensions by computing perspective or orthographic views;
10. To perform a linear mapping of points from the object's coordinate system into that of the Picture Display.

Each data coordinate that is transformed may be written into the Refresh Memory by the Terminal Control to become a portion of the new frame.

2.1.3.1.6.3 Terminal Control. The Terminal Control is the unit of the Picture Processor that controls the refresh of pictures seen on the Picture Display. The function of the Terminal Control is to receive data from the Matrix Arithmetic Processor and store it in the write portion of the Refresh Buffer. It is usually concurrently reading data from the read portion of the Refresh Buffer and sending it to the Picture Generator.

The following describes the functional specifications of the Picture Processor:

General: — The Picture Processor operations are implemented in digital hardware.

Transformations:
- — Translates objects in any direction in three space.
- — Rotates objects about any axis in three space.
- — Scales objects with respect to any of the dimensions in three space.
- — Perspective transformations can be performed on data passed to the Picture Processor.
- — The Transformation Matrix is expressed in homogeneous coordinates which allow much larger translational values than would otherwise be possible.

— Creates mirror images of objects about a plane.

Compound Transformations:
— Multiplies transformation matrices together while maintaining full-word accuracy.
— The Transformation Matrix may be loaded from the data base or stored into the data base residing in the Picture Controller memory.
— There is a push-down stack for storing four full transformation matrices with provision for continuing the stack in the Picture Controller memory.

Clipping:
— Extracts the portions of the objects, defined in the data base, that are within a program-specified field of view.
— In two dimensions, the field of view is a program-specified rectangular region of the data space.
— In three dimensions, the field of view is a pyramid or frustum (truncated pyramid) in the data space whose apex is at the eye.
— Clipping is performed with respect to the program-controlled six surfaces of the frustum.

Perspective:
— Displays realistic line representations of three-dimensional objects as they appear to the eye with reference to relative distance or depth.

Viewport:
— The viewport specification is under program control and defines a six surface region of the Picture Display where the picture is to appear.
— Data which has been transformed, clipped, and put in perspective is linearly mapped into the viewport which allows complete separation of the coordinate systems of the drawing space and the Picture Display.
— The resolution of the data mapped into the viewport is 16 bits, which allows these data to be used for precision plots.
— Multiple viewports may be defined for a given frame to give simultaneous screen.
— Specification of viewport front and back provides the intensity bounds for depth-cueing.

Zooming:
— The Picture Processor allows for moving smoothly and quickly into (or out of) a complex data structure in order to obtain a more detailed (or wide angle) view of a chosen region in the drawing space.

Hit Test:
— The Picture System can detect whether any part of a given picture element is within a program-specified region in the data space or on the
— Picture Display. Hit Test is used for implementing the pointing function with a data tablet, eliminating the need for a light pen.

Memory Write
Back: — Under program control, transformed digital data can be written back into the Picture Controller's memory to drive a hard copy plotter, for example, or as data for further computation.

2.1.3.1.6.4 The Refresh Buffer. The Refresh Buffer is a memory (distinct from the Picture Controller's) into which processed data is deposited still in digital form. This data represents the picture to be displayed on the Picture Display. For each frame refresh, the Terminal Control reads the data in the Refresh Buffer and passes the data to the Picture Generator, where the data is converted to analog signals to drive the Picture Display. Character strings from the Picture Controller pass through the Picture Processor unmodified and are deposited in the Refresh Buffer as packed character codes.

The Refresh Buffer may be operated in single or double buffer mode under program control. In single buffer mode, the entire Refresh Buffer is used to store a single display frame. In this mode, display refresh may be initiated from partially updated display frame. In double buffer mode, one half of the refresh buffer is designated as an old frame and one-half a new frame. Display refresh is then initiated from the old frame, while the new frame is being constructed. When the construction of the new frame is complete, the frame buffers are swapped and the newly constructed frame is displayed and the space occupied by the old frame becomes available for new frame construction.

The following describes the functional specification of the Refresh Buffer.

General
Function: — The Refresh Buffer stores processed digital frame data allowing complete separation of Picture Display refresh requirements from the dynamic picture update requirements.
Data Content: — Dots and line endpoint data for use by the Picture Generator (one complete dot or line endpoint definition per buffer entry containing 12 bits for each of the x and y coordinate values and 8 bits for the intensity value).
— Packed character codes for use by the Character Generator (up to three codes per buffer entry).
— Status information used to control the displaying of the data.
Buffering: — Program-selectable single or double buffering is standard.
Cursor: — A dynamic cursor can be maintained regardless of the frame update rate.

Size*: — In single buffer mode, up to 8,188 dots, line endpoints, or character code entries can be stored in the buffer in any combination.
— In double buffer mode, up to 4,092 dots, line endpoints, or character code entries can be stored in the buffer in any combination.

2.1.3.1.6.5 Character Generator. Character strings from the Picture Controller pass through the Picture Processor unmodified and are deposited in the Refresh Buffer as packed character codes. Then character words are read out of the Refresh Buffer; the Terminal Control recognizes these codes and calls upon the Character Generator to access a read-only memory containing character stroking data. The strokes are read out of the read-only memory one by one, multiplied by a pre-specified sizing parameter, and drawn by the Picture Generator on the Picture Display.

The following describes the functional specifications of the Character Generator.

General

Function: — Accepts character codes and produces properly sized digital character stroking data for the Picture Generator.

Character Set: — 96 character extended ASCII character set.

Sizes: — There are 8 character sizes available under program control ranging from 0.07 in. high in increments of 0.07 in. to 0.56 in. high on the Picture Display. The character width is also under program control with 8 different widths selectable for each size.

Character

Orientation: — Horizontal 90° counter-clockwise orientation.

Capacity: — A maximum of 1,725 characters can be displayed at a refresh rate of 30 frames per second.

2.1.3.1.6.6 The Picture Generator. The Picture Generator receives digital data consisting of x,y coordinate and intensity information read from the Refresh Memory by the Terminal Control Unit. These digital data are converted by the Picture Generator into analog signals and used to draw the picture on the Picture Display.

2.1.3.1.6.7 The Picture Display. The Picture Display receives analog signals from the Picture Generator which are used for electron beam positioning and intensity control. The

* The Standard Refresh Buffer is 8K, 36-bit words. An additional 8K of Refresh Memory may be obtained by providing a 16K Refresh Buffer.

Picture Generator controls beam positioning and the drawing of all vectors and dots on the Picture Display. The following describes the functional specification of the Picture Generator and Picture Display.

General Function: — Converts digital coordinate and intensity information to analog voltages to drive an electron beam across a phosphor-coated surface.

Line Modes:
- Solid.
- Blink mode allows selected picture elements to blink on and off.
- Dash mode allows selected lines of a picture to be dashed.

Intensity Modes:
- Constant intensity of program-selected picture elements may be chosen from 256 levels. Lines are drawn at a constant rate which assures uniform brightness for the chosen intensity level.
- Depth-cueing allows the intensity of lines to vary continuously with depth (i.e., the z coordinate of the display).

Intensity and Contrast Controls:
- In order to present a uniform variation in brightness, the intensity control of the Picture Display treats the z coordinate data as the logarithm of the intensity to be shown on the display.
- The contrast control of the Picture Display is completely independent of the intensity control.

Refresh Control: — The refresh cycle is controlled by synchronization with the power line.

Display Rates:
- Move Time (for an n in. move)
  $\leq .41 \times n + 2.85\ \mu s$ for $n \geq 1/2$ in.
  $< 3.0\ \mu s$ for $n > 1/2$ in.
- Draw Time (for an n in. line)
  $\leq 1.85 \times n + 2.0\ \mu s$ for $n \geq 1/2$ in.
  $< 3.0\ \mu s$ for $n < 1/2$ in.
- Dot Time (for dots spaced n in. apart)
  $\leq .5 \times n + 4.9\ \mu s$ for $n \geq 1/2$ in.
  $< 5.15\ \mu s$ for $n < 1/2$ in.
- Approximate display capacities at 30 frames per second refresh rate:
  1. 11,100 connected 1/2 in. lines
  2. 1,625 connected 10 in. lines
  3. 6,650 dots 1/4 in. apart
  4. 1,725 characters .14 in. high (average)
  5. 1,500 characters .56 in. high (average)

Display Type: — Calligraphic.

Deflection Type:	— Electromagnetic
Spot Size:	— 0.020 in.
Addressable Locations:	— 4,096 × 4,090.
Endpoint Matching:	— 0.020 in.
CRT Size:	— 21 in. rectangular, 10 in. × 10 in. quality viewing area.
Phosphor:	— p4.

2.1.3.1.6.8 Data Input. All data is input directly to the Picture Controller in the Picture System. Data may be input by any of the various standard peripherals available with the PDP-11 or by any of four graphical input devices supported by the Picture System:

1. Tablet,
2. Control Dials,
3. Function Switches and Lights,
4. Alphanumeric Keyboard.

The use of these standard graphical input devices provides all the capabilities normally required for graphical interaction with the Picture System.

2.1.3.1.6.9 The Tablet and Pen. The Tablet serves as the standard, general purpose graphic input device in the Picture System. The Tablet can be used for positioning or pointing to the picture elements by use of the Pen whose x,y coordinates are read by the Picture Controller. A "cursor" may be drawn on the Picture Display to indicate the position of the Pen on the Tablet. With these capabilities, the Tablet and Pen can perform the interactive functions usually reserved for such graphic input devices as light pens, joy sticks, and function switches. The Tablet is fully software-supported under the Graphics Software Package provided with the system.

The following describes the functional specification of the Tablet.

General:	— General purpose interactive input device.
Output:	— 11 bits of x, 11 bits of y, and pen up/down status.
Resolution:	— Digital: 11 bits for both x and y.
	— Graphic: 100 lines per in.
Sampling Rate:	— Variable up to 200 samples per sec.
Size:	— 11 in. x 11 in. useful area.

Cursor: — The cursor location on the Picture Display may be made to correspond to the stylus pen position on the tablet.

2.1.3.1.6.10 Control Dials. Control Dials are available with the System which permit the user to dynamically vary values which may be used to control angles of rotation, scaling factors, velocity rates, etc.

2.1.3.1.6.11 Function Switches and Lights. Function switches and lights are available with the Picture System to provide the capability for the user to utilize switches to be used for functions assigned under program control. An additional capability available with the switches is that the lights (one per switch) may be used to indicate function switch polarity or for displaying programmed information.

2.1.3.1.6.12 Alphanumeric Keyboard. The Alphanumeric Keyboard available with the Picture System is a standard 61-key, 128-character keyboard which may be used for text or data input to the Picture Controller for graphical interaction or other functions required by the user.

2.1.3.1.7 The Picture System Graphics Software Package

The Graphics Software Package furnished with the Picture System consists of a set of FORTRAN-callable subroutines written for the Digital Equipment Corporation PDP-11 computer using the MACRO-11 assembly language. These subroutines are written with the intent of providing a user with the full capabilities of the Picture System without the necessity of the user to interface on a system level with the system hardware. These subroutines provide the general user with the facilities necessary for writing interactive computer graphics programs without having to comprehensively understand the matrix arithmetic utilized within the System Processor. Instead, the user merely "calls" a subroutine to perform a required graphical function, i.e., TRANslate, ROTate, SCALE, read TABLET information, display CURSOR, display TEXT, etc.

The graphics subroutines for the Picture System have been written utilizing the PDP-11 FORTRAN calling sequence convention of the PDP-11 FORTRAN compiler V06. This calling sequence convention, supported under the DEC RT-11, DOS/BATCH, RSX-11M and RSX-11D operating systems, provides the user with flexibility of utilizing argument lists that are re-entrant or non-re-entrant in form.

All FORTRAN—callable Picture System subroutines use the standard call by name (as

opposed to call by value) parameter passing technique and specify the non-re-entrant inline form of calling sequence. Those subroutines which are not FORTRAN-callable specify no FORTRAN calling sequence.

The System Graphics Software Package may be separated into two sets of subroutines:

1. User Subroutines,
2. System Subroutines.

The User Subroutines provide all the capabilities required for the general graphical application programmer. The System Subroutines are utilized to implement the User Subroutines and are available to the programmer who wants to interface with the system software directly.

A brief description of each subroutine is contained in the following sections.

2.1.3.1.7.1 User Subroutine PSINIT. The PSINIT subroutine is called to initialize the Picture System hardware and software. The initialization process includes the following:

1. The system Real-Time Clock interrupt handler is connected to the interrupt vector and set to provide automatic refresh of the old frame and timing for frame update at the intervals specified by the calling argument list;
2. All variables are assigned their default values; all registers used in the Picture Processor are initialized for two-dimensional drawing mode; the Picture Processor is set to display data unrotated, untranslated, at full brightness, within a viewport which just fills the display screen;
3. A window is set to include the entire definition space (±32767);
4. The Refresh Buffer is set to double buffer mode with an initial frame consisting of a null cursor; the Picture Generator status is initialized to solid, 0.28 in. character size, and horizontal character mode;
5. All Picture Displays are selected for output.

2.1.3.1.7.2 User Subroutine VWPORT. The VWPORT subroutine is called to set a viewport specified by the values supplied by the operator within the calling parameters.

2.1.3.1.7.3. User Subroutine WINDOW. The WINDOW subroutine concatenates a two-dimensional or three-dimensional windowing transformation to the Picture Processor Transformation Matrix. This subroutine can be used to perform two-dimensional

windowing, orthographic projection or a true perspective transformation of data. The windowing transformation is constructed from the arguments specified in the parameter list.

2.1.3.1.7.4 User Subroutine ROT. The ROT subroutine is called to build a rotation transformation based on the angle and axis of rotation specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

2.1.3.1.7.5 User Subroutine TRAN. The TRAN subroutine is called to build a translation transformation based on the x, y, and x translational values specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

2.1.3.1.7.6 User Subroutine SCALE. The SCALE subroutine is called to build a scaling transformation based on the x, y, and z scaling terms specified in the parameter list. The transformation is then concatenated to the Picture Processor Transformation Matrix.

2.1.3.1.7.7 User Subroutine PUSH. The PUSH subroutine is called to push the current Picture Processor Transformation Matrix onto the matrix stack (hardware or memory stock, dependent upon the current stack depth).

2.1.3.1.7.8 User Subroutine POP. The POP subroutine is called to pop the top element of the matrix stack (hardware or memory stack, dependent upon the current stack depth) into the Picture Processor Transformation Matrix.

2.1.3.1.7.9 User Subroutine DRAWZD. The DRAWZD subroutine is called to draw two-dimensional data coordinate points using the drawing mode specified in the parameter list. The data to be drawn are arranged in x,y pairs and are displayed at the intensity specified by the IZ parameter.

2.1.3.1.7.10 User Subroutine DRAW3D. The DRAW3D subroutine is called to draw three-dimensional data coordinate points using the drawing mode specified in the parameter list. The data to be drawn are arranged in x,y,z triplets and are displayed at the intensity dependent upon the z coordinates and the values specified for the hither and yon planes.

2.1.3.1.7.11 User Subroutine CHAR. The CHAR subroutine is called to update the status used by the Character Generator when characters are to be displayed on the display screen.

2.1.3.1.7.12 User Subroutine TEXT. The TEXT subroutine is called to display the text string specified in the parameter list. The display of the text will be from the current beam position and at the intensity associated with the last information displayed. The character status will be initialized by PSINIT or updated by the CHAR subroutine if previously called by the user.

2.1.3.1.7.13 User Subroutine INST. The INST subroutine concatenates a two-dimensional or three-dimensional instancing transformation to the Picture Processor Transformation Matrix. This subroutine is used in conjunction with the MASTER subroutine to produce multiple instances of an object or symbol. For each desired appearance of the object, the INST subroutine is called to specify the location (and implicitly the size) of that appearance; then the user-supplied routine describing the object is called to display the object previously defined within a two-dimensional or three-dimensional enclosure. The INST subroutine pushes the initial Transformation Matrix onto the Transformation Stack before concatenating the instancing transformation, so that it may be restored (transferred) by the user after the object has been drawn.

2.1.3.1.7.14 User Subroutine MASTER. The MASTER subroutine concatenates a two-dimensional or three-dimensional master transformation to the Picture Processor Transformation Matrix. This subroutine is used in conjunction with the INST subroutine for instancing of data. The master transformation is constructed from the arguments specified in the parameter list.

2.1.3.1.7.15 User Subroutine DASH. The DASH subroutine is called to set the Picture Generator status such that all subsequent lines drawn will be dashed or nondashed dependent upon the value of the argument.

2.1.3.1.7.16 User Subroutine BLINK. The BLINK subroutine is called to set the Picture Generator status such that all subsequent lines drawn will or will not blink, dependent upon the value of the argument. Data drawn in Blink mode will blink at approximately 90 blinks per minute.

2.1.3.1.7.17 User Subroutine SCOPE. The SCOPE subroutine is called to select the Picture Display to which output will be directed.

2.1.3.1.7.18 User Subroutine TABLET. The TABLET subroutine is called to read the current pen position and status in relation to the tablet. The user may also specify initiation of automatic tablet mode. This will cause the current pen position to be updated at each

frame refresh. This ability, used in conjunction with the automatic cursor mode, allows completely dynamic cursor tracking irrespective of new frame update rate. It should be noted that once the pen information is updated with the pen down bit set, the pen position will not be updated until the user has cleared the pen value word indicating that the pen down position has been read or until the pen is set down again.

2.1.3.1.7.19 User Subroutine ISPDWN. ISPDWN (IS PAN DOWN) is a FORTRAN-callable integer function subroutine which may be used to determine whether the pen is down (i.e., *pressed* against the surface of the tablet). This function routine allows FORTRAN applications programs, which do not have the ability to perform bit testing, to test the pen up/down status.

2.1.3.1.7.20 User Subroutine CURSOR. The CURSOR subroutine is called to display a cursor at the position specified by the parameter list. This will cause a cursor to be displayed upon each frame refresh irrespective of the new frame update rate. The cursor displayed in automatic cursor mode will be at the position specified by the x and y position values and within the viewport that had been specified at the time of the initial CURSOR call. The cursor displayed consists of a cross with a center at the x and y position specified.

2.1.3.1.7.21 User Subroutine HITWIN. The HITWIN subroutine is called to specify a window through which data will be passed to determine whether data are being drawn within a given area. The user specifies an x and y coordinate where a window transformation of the specified size is centered. This window transformation is then pre-concatenated with the transformation in the Picture Processor Transformation Matrix, after first saving the original transformation so that it may be restored after all hit testing has been completed. The Picture Processor status is then set to prohibit all data drawn from being output to the Refresh Buffer. The subroutine then returns to allow the user to draw all data against which hit testing is to be performed.

2.1.3.1.7.22 User Subroutine HITEST. The HITEST subroutine is called to determine if any output data has passed within a prespecified hit window (see HITWIN). The procedure for this test is of the form:

1. CALL HITWIN to set up the desired hit window,
2. Draw data (DRAW2D and/or DRAW3D) for comparison against that window,
3. CALL HITEST to determine if there was a "hit",
4. Repeat steps two and three as often as necessary, setting HITEST argument two to a non-zero value on the last call to HITEST to restore the former user transformation.

2.1.3.1.7.23 User Subroutine NUFRAM. The NUFRAM subroutine is called to initiate the switch from displaying the old frame data to displaying the new frame data (the actual frame switch does not occur until the appropriate refresh interval).

2.1.3.1.7.24 User Subroutine SETBUF. The SETBUF subroutine is called to set the Refresh Buffer to single-buffer or double-buffer mode. Once the Refresh Buffer has been set to a mode, it may be reset at any time to the other mode. The user needs to call this subroutine only if the Refresh Buffer is used in single buffer mode. PSINIT during the initialization process sets the Refresh Buffer to the default double-buffer mode.

2.1.3.1.7.25 User Subroutine PSWAIT. The PSWAIT subroutine is called whenever it is necessary to wait until the Picture Processor and Direct Memory Access Unit have completed their present operations before continuing. This is used to insure that the data transfer to or from the Picture Controller's memory is complete before the data is referenced or modified.

2.1.3.1.7.26 System Subroutine BLDCON. The BLDCON subroutine is called to perform all transformation operations and matrix manipulations.

2.1.3.1.7.27 System Subroutine P$AVE. The P$AVE subroutine is called to save registers R0-R5 on the program stack.

2.1.3.1.7.28 System Subroutine R$TORE. The R$TORE subroutine is called to restore registers R0-R5 from the program stack.

2.1.3.1.7.29 System Subroutine P$DMA. The P$DMA subroutine is called to initiate a Direct Memory Access (DMA) transfer and check for the correct completion of the operation.

2.1.3.1.7.30 System Subroutine I$MATX. The I$MATX subroutine is called to initialize a 16-word array in memory (P$MATX) to a 4 x 4 identity matrix.

2.1.3.1.7.31 System Subroutine ERROR. The ERROR subroutine is called by all Picture System subroutines that encounter an error condition during the course of execution. This subroutine in turn calls the user error subroutine specified in the call to PSINIT or the default system error routine.

The following two function subroutines are optimized for the particular PDP-11 hardware configuration.

2.1.3.1.7.32 Function Subroutine P$DIV. The P$DIV function subroutine divides the signed division in R0 and R1 by the signed divisor in R2, leaving the quotient in R0 and the remainder in R1 with R2 undisturbed. The quotient bears the algebraic sign of the division, while the remainder retains the sign of the dividend.

2.1.3.1.7.33 Function Subroutine P$MUL. The P$MUL function subroutine multiplies the signed multiplicand in R0 by the signed multiplier in R2, leaving a signed product in R0 and R1 with R2 undisturbed.

2.1.3.1.8 Picture System errors

Error detection by the Graphics Software Package is performed to insure program integrity and to facilitate program debugging. A user may make four types of programming errors that will be detected by the Graphics Software Package. These are:

1. The call of a graphics subroutine with an invalid number of parameters specified,
2. The call of a graphics subroutine with an invalid parameter value,
3. The attempt by the user to PUSH the matrix stack to a depth greater than that specified by the user in the call to PSINIT,
4. The attempt by the user to POP a transformation from the matrix stack which had not been previously PUSHed.

When an error is detected by a graphics subroutine, the system subroutine ERROR is called with an argument that specifies the origin of the error detected and the error condition encountered. The system subroutine ERROR then calls the user error subroutine, specified in the call to PSINIT. When called, a parameter will be passed by the user error subroutine, which specifies the origin and type of error detected. Return from the user error subroutine will result in the termination of the program. If, in the call to PSINIT, the user does not specify an error subroutine, the graphics error subroutine PSERRS will be called. PSERRS, when called, will be called. PSERRS, when called, will output the following message to the console terminal. . .

ERROR X DETECTED IN GRAPHICS SUBROUTINE YY.

. . . and terminate the execution of the program.

Because a comprehensive set of system diagnostics is provided with the Picture System,

hardware error detection is performed to a minimal level. There are, however, two error codes which may indicate a hardware failure. They are:

1, 2. Direct Memory Access Error; This indicates that an error occurred during the last Direct Memory Access operation;

1, 3. Matrix Stack Error; This error indicates that the Picture Processor detected an attempt to overflow or underflow the hardware matrix stack; this error can only be caused by a hardware malfunction or by a system programming error, and if no software error is apparent, the PUSH/POP diagnostic routine may be run to check the integrity of the hardware.

2.1.3.2 Video Control Center

The purpose of the Video Control Center is to generate, control, monitor, and record all standard video signals needed in the AVSAIL simulation facility. Video signals may be generated live, from video tape, or from movie film. The video signals can be monitored at the console and distributed to a remote location such as the cockpit simulator. Special effects can be used to alter the signal. Cameras are available to observe the pilot's movements during a simulation. This information, as well as all other video signals, can be recorded on tape for future reference. The system can also serve as a training tool in that demonstration and instruction tapes can be produced.

The Flying Spot Scanner is designed to simulate a raster type sensor on an aircraft. The scanner is included in this report because it is a source of video and depends upon control signals from the Video Control Center for operation. A photograph of the complete console (with the exception of the Flying Spot Scanner) is shown in Figure 2.1.3.2-1.

2.1.3.2.1 Video console

Figure 2.1.3.2-2 is a sketch of the console with the various functional components numbered. A listing of these components is given in Table 2.1.3.2-1. Each component is described briefly in the following sections.

A standard video signal has three main parts: SYNC, BLANKING, and VIDEO INFORMATION. Figure 2.1.3.2-3 defines these and other terms used throughout this discussion.

2.1.3.2.1.1 AC Power. Nearly all pieces of equipment in the console are powered

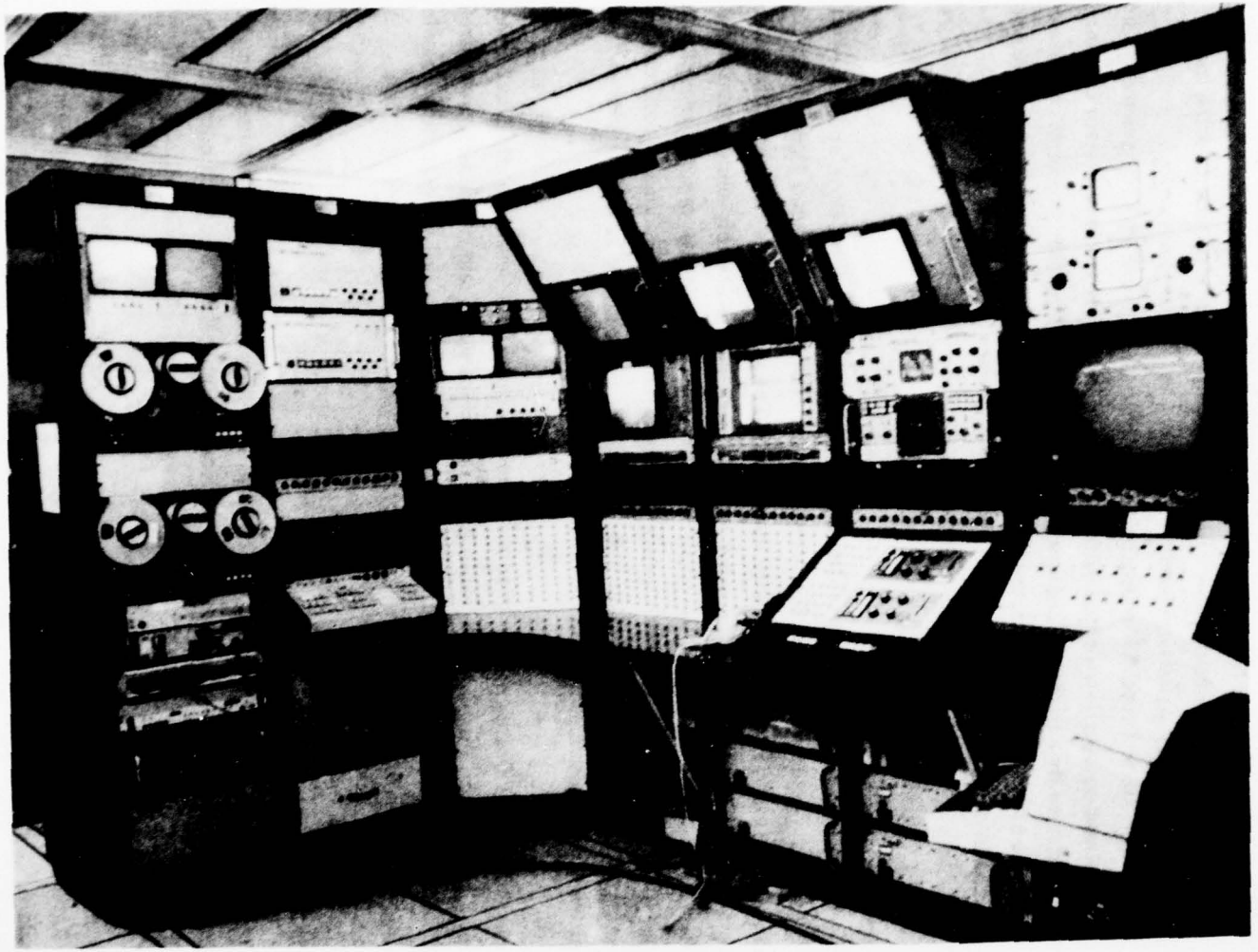


Figure 2.1.3.2-1. The Video Center.

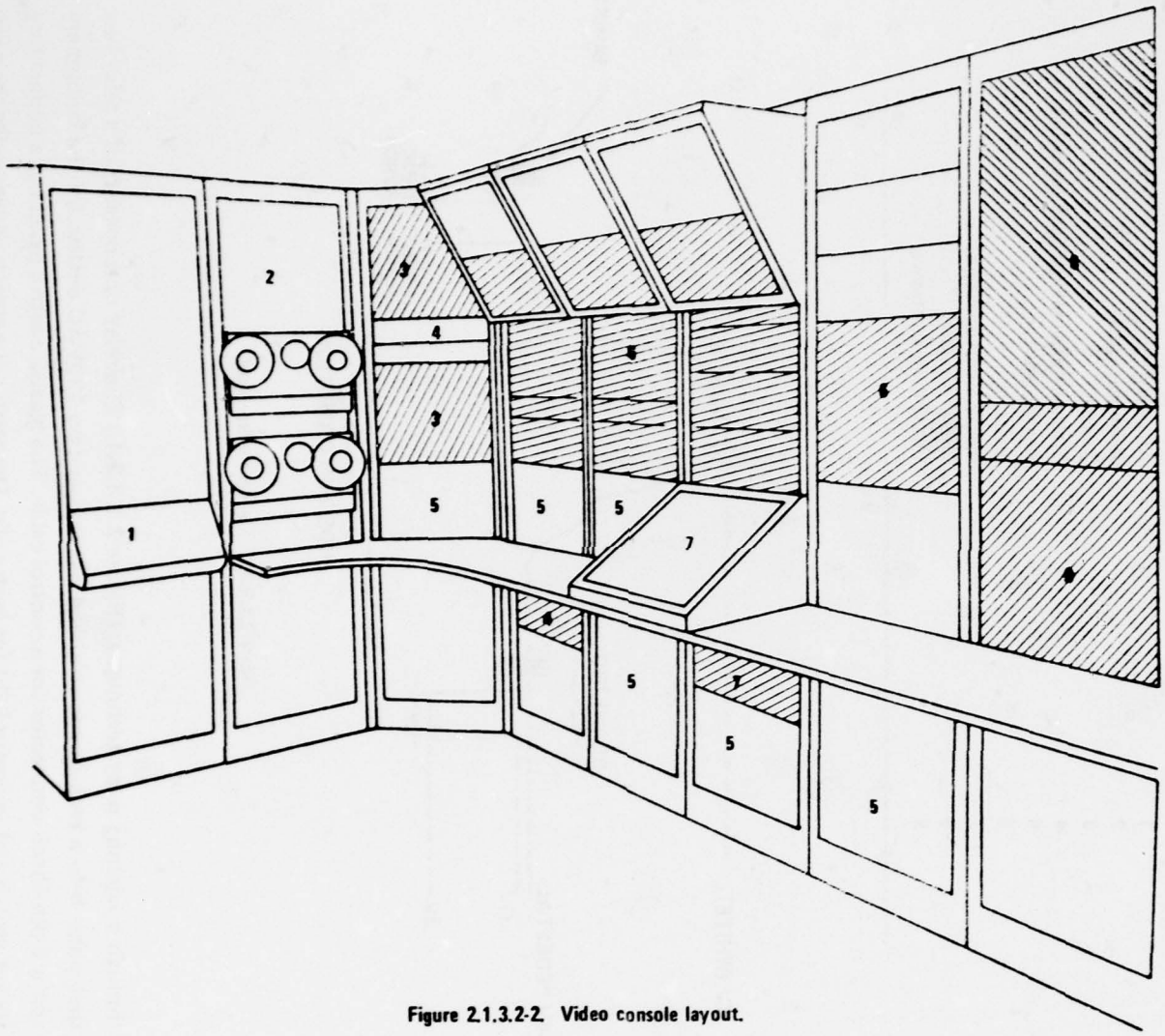


Figure 2.1.3.2-2. Video console layout.

TABLE 2.1.3.2-1. VIDEO CONSOLE COMPONENTS

Console Component Number	Console Components
1	AC power
2	Video tape system
3	Cameras
4	Synchronization and test signal connectors
5	Routing switcher
6	Monitors
7	Special effects generator
8	PDP-11 interface
9	Part of flying spot scanner

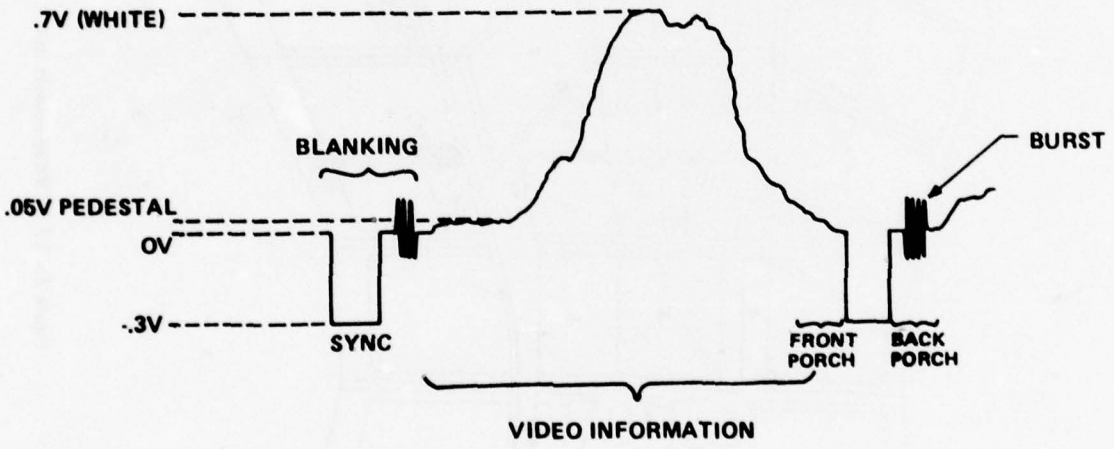


Figure 2.1.3.2-3. Standard video signal.

through a latching relay technique. (Figure 2.1.3.2-4.) The relay unit consists of a relay box that plugs into a rack power strip and has one or two 120VAC outlets for the equipment and a 3-pin Cinch connector for a control cable. The master control panel has a momentary rocker switch and indicator light for each relay. Depressing the switch will energize the relay and cause it to switch states. A second energizing will cause the relay to switch back (latching impulse relay). The indicator light shows the state of the relay box output, hence the status of the equipment connected to that box.

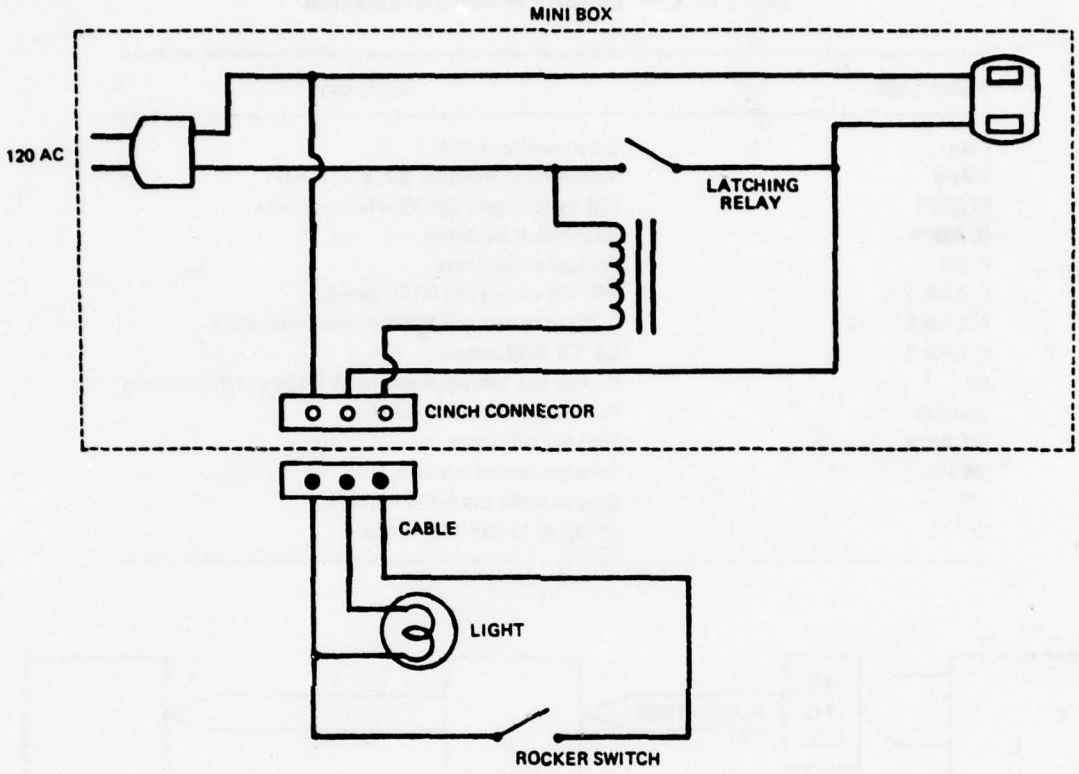


Figure 2.1.3.2-4. Power control.

The various equipments powered from the switch panel are listed in Table 2.1.3.2-2.

Not all equipment power is controlled by these relays. Such items as the Time Base Correctors which have obvious power switches and pilot lights are not controlled in order to conserve relays. Other items such as the 146 SYNC Generator are on continuously for frequency stability.

2.1.3.2.1.2 Routing Switcher. The routing switcher is the heart of the video system. (Figure 2.1.3.2-5, -6.) The 400 crosspoint video switcher selects and distributes video to all destinations.

The purpose of the switcher is to perform a function equivalent to connecting coax from a video source to a destination by merely pushing a button. The system also provides a readout tally of completed circuits.

TABLE 2.1.3.2-2. CONSOLE POWER DISTRIBUTION

Switch Label	Equipment
CMx	Color monitor x (0-4)
BMx-y	Monochrome monitors 5-7, 8-10, 11-12
520/529	520 Vectorscope, 529 Waveform monitor
D AMPS	Distribution amplifiers
P SW	Production switchers
C CAM 1	IVC 150 camera and NTSC encoder
C CAM 2	IVC 928 camera and NTSC encoder (film chain)
C CAM 3	GE TE 1000 camera
VAE	Vertical aperture equalizers for IVC-150 and 928 cameras
Switcher	Main power to routing switcher
SW lights	Switcher tally lights
SW +5	Switcher control interface
±15	Scanner raster and PDP-11 interface
INT +5	+5 supply to PDP-11 interface

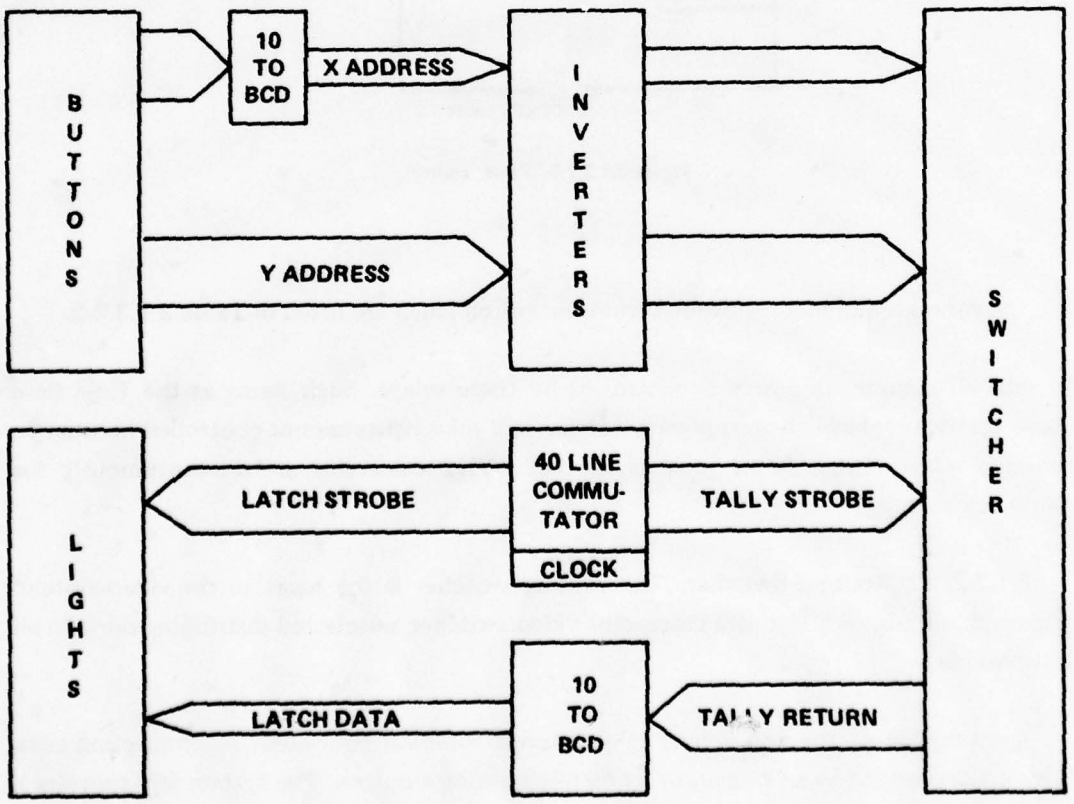


Figure 2.1.3.2-5. Routing switcher.

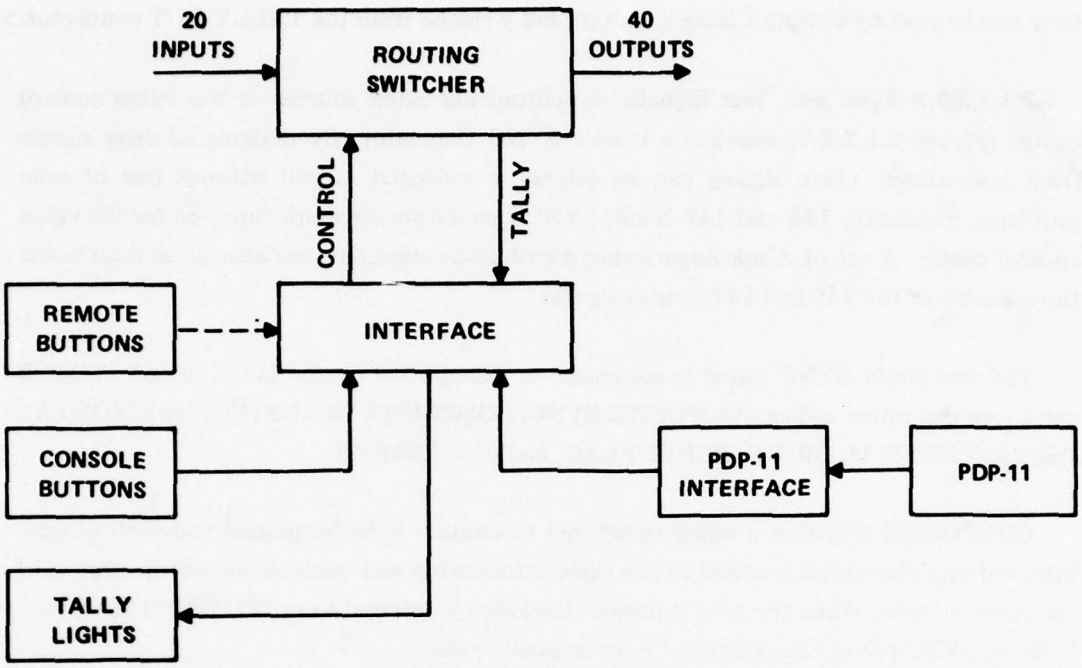


Figure 2.1.3.2-6. Routing switcher.

The control inputs require TTL compatible signals whereas the tally-out signals are discrete transistors. The interface circuits act as a buffer/summer for incoming commands and a data multiplexer for returned tally information. The buffer input is designed to accept commands from more than one source such as from console push-buttons, remote push-buttons or the PDP-11.

The switcher's 400 crosspoints are arranged as 10 inputs and 40 outputs. This is true to the extent that a given output can select any one of 10 inputs. However, the system is provided with 20 inputs. By means of internal jumper wires, each output can be switched between a selected 10 of the 20 inputs. An attempt was made to provide an output with only the inputs logically required.

Control from the push-button panels of the console requires that one simply locate the row of buttons corresponding to the desired output and press the button labeled with the desired input.

The switcher control input requires two addresses in TTL compatible form. The x address is a complimented 4-bit BCD digit (0-9) corresponding to the input desired. The y address is a discrete 1 of 40 strobe to designate the output. (Refer to Figure 2.1.3.2-5.) The

tally can be read by sensing a BCD x address and y strobe from the TALLY OUT connector.

2.1.3.2.1.3 Sync and Test Signals. Synchronizing video sources in the video control center (Figure 2.1.3.2-7) requires a master SYNC Generator. By deriving all drive signals from one source, video signals can be mixed or switched at will without fear of sync problems. Tektronix 146 and 147 NTSC SYNC Generators serve this function for the video control center. A set of Alma Engineering distribution amplifiers are also present to boost the capacity of the 146 and 147 output signals.

The maximum SYNC signal requirement is exemplified by the IVC-150 and IVC-92B color cameras which utilize COMPOSITE SYNC, COMPOSITE BLANKING, HORIZONTAL DRIVE, VERTICAL DRIVE, BURST FLAG, and SUBCARRIER.

COMPOSITE SYNC is a signal structured to contain both horizontal and vertical sync information. This signal is added to the video information and controls the sweep circuits of the video receiver. When the sync is present the video is referred to as "COMPOSITE video". Without SYNC present the signal is "noncomposite video".

COMPOSITE BLANKING, like COMPOSITE SYNC, contains both vertical and horizontal BLANKING information. Since the purpose of BLANKING is to blank out video information during the retrace periods, COMPOSITE BLANKING is not used to control sweep circuits as is COMPOSITE SYNC.

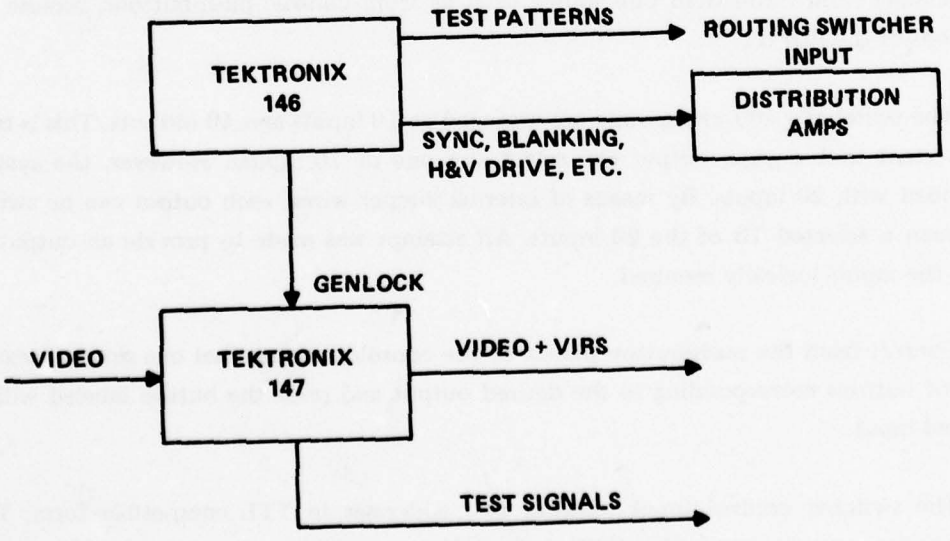


Figure 2.1.3.2-7. SYNC and test signals.

HORIZONTAL and VERTICAL DRIVE signals have a duty cycle that can control the trace and retrace times of the horizontal and vertical sweeps respectively. In the case of most video sources (cameras, etc.), H and V DRIVE controls the sweep circuits, while COMPOSITE SYNC is merely added to the output signal. In the case of the TE-26 cameras, removing the COMPOSITE SYNC input only changes the output from "composite" to "noncomposite" but does not disable the camera.

SUBCARRIER is a 3.579545 MHz signal which is the basis for NTSC encoded color transmission. The SUBCARRIER output from the SYNC Generator is a highly stable source used to phase-lock the internal oscillators of the NTSC encoders. By having one source of subcarrier phase reference, all video sources can be color synchronized and sweep synchronized.

The video receiver reference is the BURST which is located in the "back porch" of the signal. The BURST FLAG is a signal which acts as a gate for insertion of the BURST into the video signal.

In addition to the SYNC and DRIVE signals, the generators also provide several test signals. Some examples are COLOR BARS, CROSSHATCH, MULTIBURST, and CALIBRATED NOISE. These signals are primarily used for calibrating, aligning, or evaluating video transfer circuits or receivers.

The 147 has another feature in that it can insert on a video signal the Vertical Interval Reference Signal (VIRS). This signal is a line of video occurring during the vertical blanking period, which has a standard LUMINANCE AND CHROMA content. After the VIRS is added, the signal may pass through several amplifiers, distribution systems, or transmitters and receivers, each of which could alter the signal characteristics. By examining the VIRS at the final destination and processing the signal so as to restore the VIRS signal to its standard characteristics, the remainder of the signal is corrected as well. A device that performs this processing is the Tektronix 1440.

2.1.3.2.1.4 Monitors. Three general classes of monitors (Figure 2.1.3.2-8) are present in the system—MONOCHROME, COLOR, and WAVEFORM. The color monitors in the console are standard NTSC units with 12-in. screens. The Tektronix 650 is a much more precise unit than the other monitors and is used for all critical examination work. All color monitors have a cross-pulse feature which allows viewing of the SYNC and BLANKING information normally occurring during retrace. It is thus possible to examine for SYNC stability, video tape alignment and drop out, or presence of BURST.

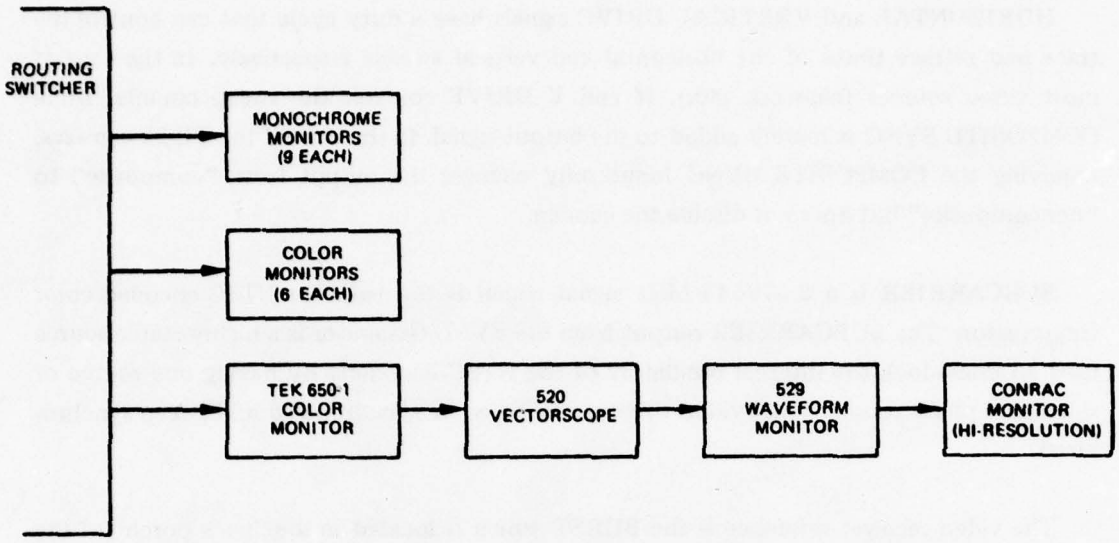


Figure 2.1.3.2-8. Monitor system.

The 650 is connected or "looped" with a Tektronix 520 Vectorscope and a 529 Waveform Monitor. Routing a signal to CM4 will also route the signal to the 520 and 529. The 529 operates much as an oscilloscope with sweep and trigger circuits "tuned" to video line and field rates. In the line rate mode, the display shows lines of video with the SYNC, BLANKING, and BURST visible. In the field mode, an entire field of video is displayed. The 529 Waveform Monitor is also useful in aligning cameras.

Waveform examination can be aided by using the input filters provided on the 529. FLAT response allows the entire signal to pass to the screen. HIGH PASS essentially passes only the subcarrier or CHROMA information. IEEE and LOW PASS are both low pass filters, but the IEEE filter has a much sharper cutoff and serves to remove the CHROMA information, whereas the LOW PASS filter will distort the SYNC as well.

The 520 Vectorscope is primarily for use on NTSC color signals. It will display line information but not the SYNC interval. The Y, R, G, and B buttons select a line display of the separate LUMINANCE, RED, GREEN, and BLUE information. The 520 does not, however, display the actual subcarrier as the 529 does. The 520 decodes and separates the color signal before displaying.

The remaining category of monitors is the MONOCHROME category. The console has several small monitors which simply display monochrome video and have no features such as

cross-pulse. One large-screen MONOCHROME monitor exists for special purposes. The Conrac RQA-17 is designed to automatically lock to any line rate from 450 to 1300 lines/frame. It is wired to the 650/520/529 monitor loop.

2.1.3.2.1.5 Cameras. The following cameras (Figure 2.1.3.2-9) are available in the video system:

1. IVC-150B
2. IVC-92B
3. TE-1000
4. TE-26B
5. TMC-2300
6. Color

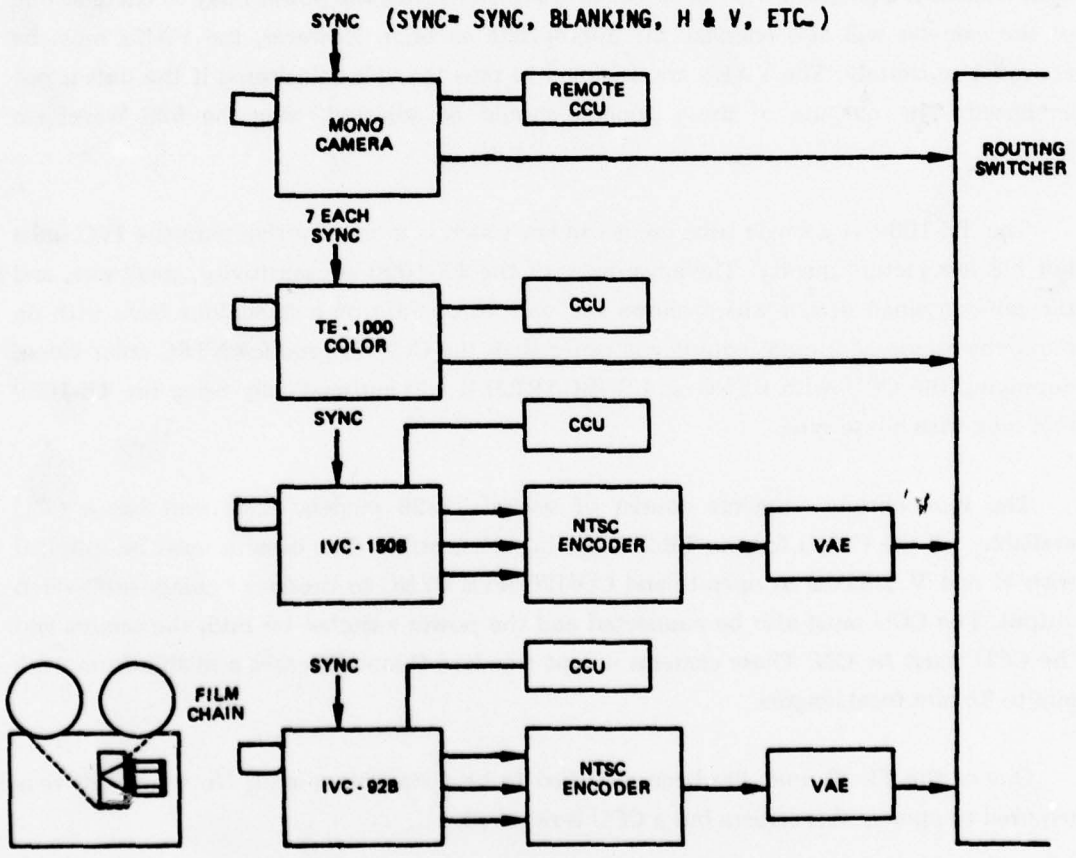


Figure 2.1.3.2-9. Camera system.

7. Color
8. Color
9. Monochrome
10. High Resolution Monochrome

The IVC-150 is a three tube, studio grade, color camera. It features a 10:1 zoom lens and remote iris controllable from the Camera Control Unit (CCU) located in the console. This camera is useful for taping demonstrations or for the generation of any high quality, live color video. The IVC-92B is another three tube color camera which is dedicated to the Film Chain system. In place of the iris control the 92B has a Video Gain control in the CCU. Associated with each of these cameras is a Vertical Aperture Equalizer (VAE) which processes the NTSC video and enhances edges and contours to give a sharper picture.

The two cameras output RGB color video rather than NTSC encoded video. Therefore, each camera is associated with an NTSC encoder. Activating the power relay to energize one of the cameras will also energize the appropriate encoder. However, the VAE's must be activated separately. The VAE's are designed to pass the video unaltered if the unit is not energized. The outputs of these cameras should be adjusted using the 529 Waveform Monitor.

The TE-1000 is a single tube color camera which is more sensitive than the IVC units but has less picture quality. The advantages of the TE-1000 are sensitivity, small size, and the self-contained design which allows the unit to operate on a standalone basis with no requirement for additional equipment other than the CCU to produce NTSC color video. Supplying the CCU with SYNC and SUBCARRIER will automatically bring the TE-1000 into lock with house sync.

The monochrome cameras consist of several TE-26 models. Each unit has a CCU available. All the CCU's for the TE-26's are interchangeable. The camera must be supplied with H and V DRIVE to operate and COMPOSITE SYNC to produce "composite" video output. The CCU must also be connected and the power switches on both the camera and the CCU must be ON. These cameras accept standard C-mount lenses available from 12.5 mm to 75 mm focal lengths.

One of the TE-26 units has been modified to be a standalone unit. No sync or drive is required to operate this camera but a CCU is required.

The remaining monochrome type camera is the TMC-2300, 1000-line. This model is a standalone unit. It is designed to operate as a totally self-contained camera head or to

operate with a CCU (located in the console). The only monitor presently available for 1000-line video is the Conrac RQA. The 2300 has additional features of scan reversal and video inversion. The 2300 also accepts standard C-mount lenses.

A set of neutral density filters is installed between the 92B and the optical multiplexer to control the intensity of light entering the camera. A control on the filter unit varies the density of the filters. The IVC-150 and 92B each have an electronic viewfinder in the form of a small monochrome monitor located in the back of the camera.

2.1.3.2.1.6 Video Tape System. The Video Tape System (Figure 2.1.3.2-10) consists of two IVC-870, two IVC-800 helical scan Video Tape Recorders (VTR), and two CVS-504

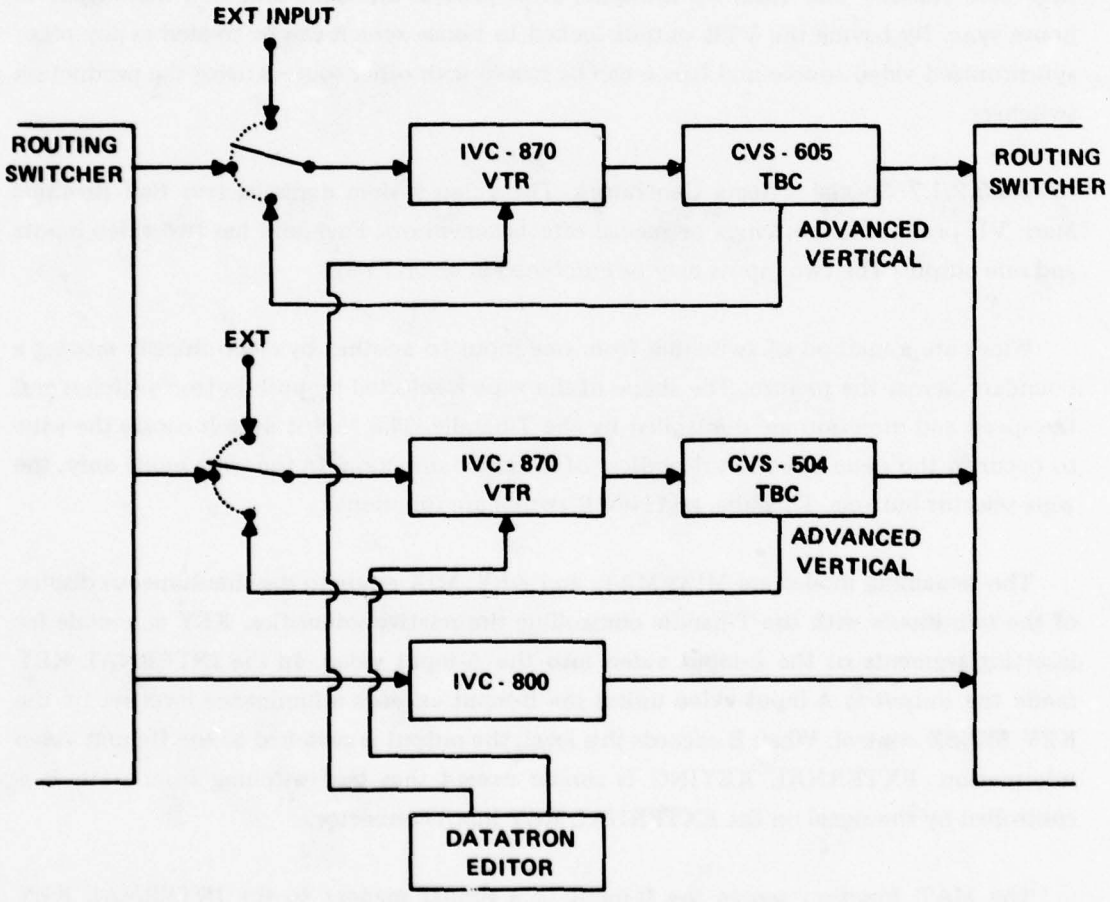


Figure 2.1.3.2-10. Video tape recorder system.

Time Base Correctors (TBC). One 800 is a portable machine usually divorced from the system. This machine can be connected using the VT3 input and output of the routing switcher. The other 800 is slated for use in a portable console for taping lectures or demonstrations physically inconvenient to the console.

The 870's and the TBCs operate together to form the main Video Tape System. Inputs to the VTRs are controlled by the VT1 and VT2 outputs of the routing switcher. The TBC can either pass the VTR output unaltered and process the signal for time base stability, or process and lock the signal to house sync.

To play back, three options exist. Unprocessed video can be viewed without using the TBC (monochrome only). The TBC can be used to either process the signal and improve time base stability and clean up the sync, or to process the video and lock the output to house sync. By having the VTR output locked to house sync it can be treated as any other synchronized video source and hence can be mixed with other sources using the production switchers.

2.1.3.2.1.7 Special Effects Generators. The video system contains two Ball Brothers Mark VII production switchers or special effects generators. Each unit has two video inputs and one output. The two inputs may be combined in several ways.

Wipes are a method of switching from one input to another by electronically moving a boundary across the picture. The shape of the wipe is selected by push-button switches and the speed and direction are controlled by the T-handle. The N-N/R switch causes the wipe to occur in the same direction regardless of T-handle direction. In the wipe mode only, the wipe selector buttons, T-handle, and N-N/R switch are functional.

The remaining modes are MIX, MAT, and KEY. MIX refers to the simultaneous display of the two inputs with the T-handle controlling the relative intensities. KEY is a mode for inserting segments of the B-input video into the A-input video. In the INTERNAL KEY mode the output is A-input video unless the B-input exceeds a luminance level set by the KEY SENSE control. When B exceeds this level, the output is switched to the B-input video information. EXTERNAL KEYING is similar except that the switching from A to B is controlled by the signal on the EXTERNAL KEY input connector.

The MAT function senses the B-input in a similar manner to the INTERNAL KEY function. Rather than switching from the A to the B input however, the output is switched from the A-input to a solid color as determined by the HUE, SATURATION, and

LUMINANCE controls. When used with the MIX mode on the second unit, a slight tint can be added to a picture.

The outputs of the two production switchers are designated PS1 and PS2. Four sections of the routing switcher control the inputs of the production switchers.

2.1.3.2.2 Flying Spot Scanner

The Flying Spot Scanner (FSS) is a special video source system consisting of several parts. The general classifications are: Raster Generator/Processor, Deflection Controller, Position Controller, Video Detector, and Video Processor.

In relation to theory of operation, the main parts are: CRT (Optics), Transparency, and Photomultiplier Tube (PMT). The CRT displays a scan of the desired shape with no intensity modulation. The optic system focuses the CRT scan on the transparency.

A PMT is located behind the transparency to convert the instantaneous light intensity to a voltage waveform. In this manner, the phosphor surface of the CRT is equivalent to the surface of the transparency. Thus, scanning the CRT is equated to scanning the transparency. The output of the PMT is a voltage waveform in time-sync with the CRT scan. Additionally, the optics included usually reduce the scan size to increase resolution. By changing the time synchronization between the input CRT scan and the photomultiplier tube output, the apparent view of the transparency can be changed.

The sweeps for the CRT, Hs, and Vs are derived from a Celco (Constantine Engineering Laboratories) raster generator which accepts H and V DRIVE from the 146 SYNC generator and produces two ramp voltage waveforms equivalent to a 4:3 aspect raster.

The input control voltages can be derived from two sources. For manual operation, a set of controls is located near the Raster Processor. The voltmeter can be switched to monitor any one of the control voltages. Throwing the toggle switch below a control knob will revert control of that function to a Digital to Analog Converter (DAC) driven by the PDP-11 interface. In this manner, any combination of computer and manual control is possible.

2.1.3.2.2.1 Raster Generator/Processor. For this FSS system, the desired output is standard 525/60 video. Therefore, the input scan must be a set of waveforms time-synced to 525/60 rates. Beginning with two orthogonal Hs and Vs sweeps from a raster generator, the Raster Generator/Processor alters the sweep waveforms to achieve the effect of a TV sensor on an aircraft.

2.1.3.2.2.2. Deflection Controller. The Deflection Controller section contains the Deflection Drivers, CRT, and CRT Power Supplies. Activating the main power control for the Video Cabinet will begin a turn-on sequence controlled by time-delay relays. The order of activation is as follows: Deflection Drivers and PMT Low Voltage, CRT Gun Supply, PMT High Voltage, CRT High Voltage and Focus. A set of indicator lights track the progress of the five-minute power-up cycle. If any of the scanner door interlock switches are tripped, the CRT High Voltage will not turn on, and an indicator will light showing the interlock failure. When the failure is corrected, the light will extinguish but the High Voltage supply will not activate for 30 seconds.

The sweep signals from the Raster Generator/Processor provide the input for the Deflection Drivers. The driver unit is a Celco RDA-1,260 dual axis driver connected to a Celco Deflection precision yoke. The RDA-1,260 is a 60-volt, 12-amp system which can fully deflect the beam at a rate of 30 KHz.

In magnetic deflection systems the current through the yoke determines the beam deflection. A constant current will cause a fixed deflection from center. The RDA-1,260 is essentially two high power, operational amplifiers which convert an input voltage to a proportional current through the yoke. The input voltage waveform is duplicated in the output current waveform on a one volt equals one amp basis.

2.1.3.2.2.3 Cathode Ray Tube (CRT) System. (Figure 2.1.3.2-11.) The CRT is a seven-in., flat screen, 42° deflection tube with a 1.5 mil minimum spot.

The light from the CRT is focused by mirrors and lenses onto the surface of the transparencies. Two 20 x 20 in. glass transparencies are mounted in a single carrier frame. The light from the CRT is split into two beams such that the two transparencies are scanned in parallel. A photomultiplier tube for each plate is located on the opposite side from the CRT and optics. The carrier frame is positioned by a set of X-Y servos.

The CRT is driven by the CRT Gun Supply and High Voltage Supply located in the Video Cabinet. The Gun Supply is a Litton model 1050 which supplies the filament and grid voltages to the CRT.

The 1050 Gun Supply also has a CRT intensity modulator. The modulator circuit board has been removed from the 1050 chassis and mounted adjacent to the CRT in the Scanner Cabinet.

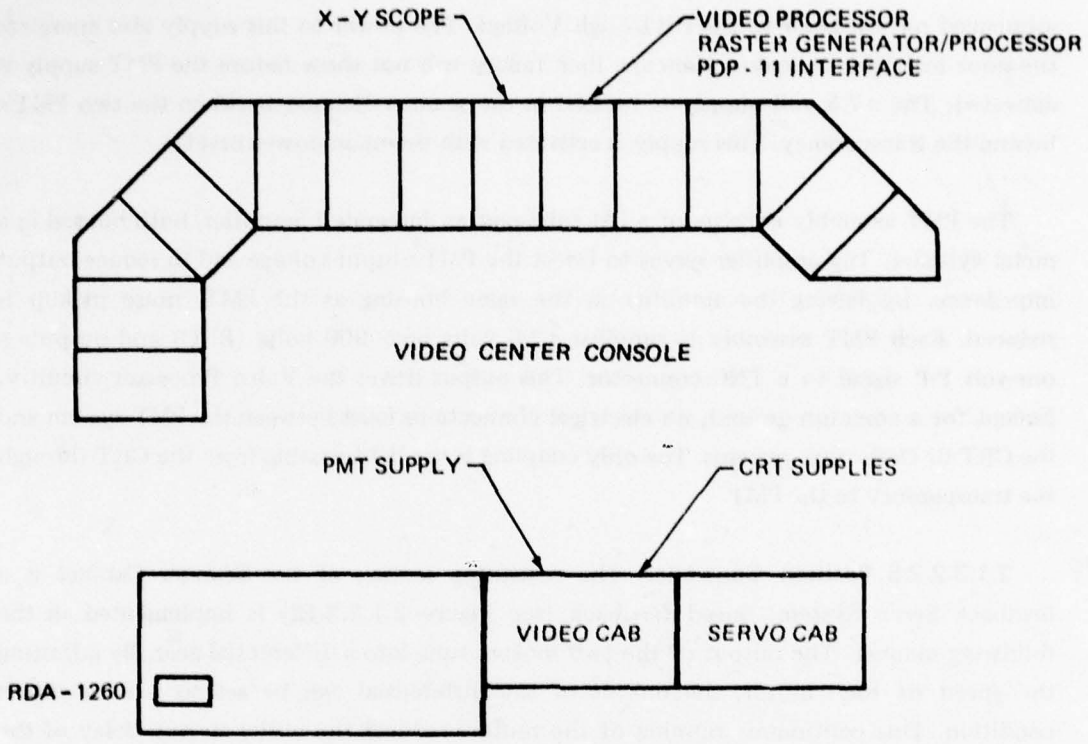


Figure 2.1.3.2-11. CRT system.

The modulator is driven by the Video Processor to blank the CRT during retrace and periods corresponding to horizon and sky of the generated scene. The details of this circuit are explained in the section on the Video Processor.

The focus supply is a Litton 1008 for magnetic focus systems. The focus coil is mounted behind the deflection yoke on the CRT. The focus control is located on the 1008 in the Video Cabinet. The High Voltage supply is a Spellman RG-30, 30 kv supply. The normal setting for this system is 27 kv. Since the RG-30 is a vacuum tube supply, approximately 15 seconds is required for warm-up. The final sequence relay activates both the 1008 and the RG-30. Both supplies drop out if a door interlock switch is tripped. Normally these supplies are not adjusted during operation.

2.1.3.2.2.4 Video Detector. The photomultiplier tubes require +7.5 volts and -900 volts. A Kepco supply located in the Video Cabinet provides the 900 volts. This supply is

sequenced on just prior to the CRT High Voltage. The power to this supply also energizes the door interlock indicator. Hence, a lock failure will not show before the PMT supply is activated. The ±7.5 volt supply is located in the scanner Cabinet between the two PMT's behind the transparency. This supply is activated with the main power breaker.

The PMT assembly consists of a PM tube and an integrated amplifier, both housed in a metal cylinder. The amplifier serves to boost the PMT output voltage and to reduce output impedance. By having the amplifier in the same housing as the PMT, noise pickup is reduced. Each PMT assembly is supplied ±7.5 volts and -900 volts (BNC) and outputs a one-volt P-P signal to a TNC connector. This output drives the Video Processor circuitry. Except for a common ground, no electrical connections exist between the PMT system and the CRT or Deflection systems. The only coupling is the light passing from the CRT through the transparency to the PMT.

2.1.3.2.2.5 Position Controller. The remaining section of the Scanner Cabinet is a feedback Servo System. Speed feedback (see Figure 2.1.3.2-12) is implemented in the following manner. The output of the two motors runs into a differential gear. By adjusting the speed of each motor, the output of the differential can be set to a nonrotation condition. This continuous running of the motors reduces the initial startup delay of the servos. When the course position gets within a certain window the analog switch changes the motor speed control from coarse control to fine control.

The present system is interfaced through digital to analog converters to the PDP-11.

2.1.3.2.2.6 Video Processor. The output from the photomultiplier tube system is a raw video signal with no SYNC, BLANKING, or black reference level. The Video Processor section contains the circuitry to clean up the video and add the necessary SYNC and some special effects. (Refer to Figure 2.1.3.2-13.)

The Proc Amp circuitry receives the raw video from the PMT and performs the following functions. The signal is keyclamped and limited to produce a uniform amplitude, unipolar signal, meeting "video information" signal specifications. The Proc Amp then blanks the signal and inserts SYNC as required. BURST is also inserted in the "back porch". As shown in Figure 2.1.3.2-13, the Proc Amp uses signals from the Tektronix 146 to perform these tasks. The output from the Proc Amp is a standard NTSC video signal.

In addition to the above tasks the Proc Amp has the feature of adding a CHROMA signal to the video. This CHROMA signal is generated by the Horizon Simulator.

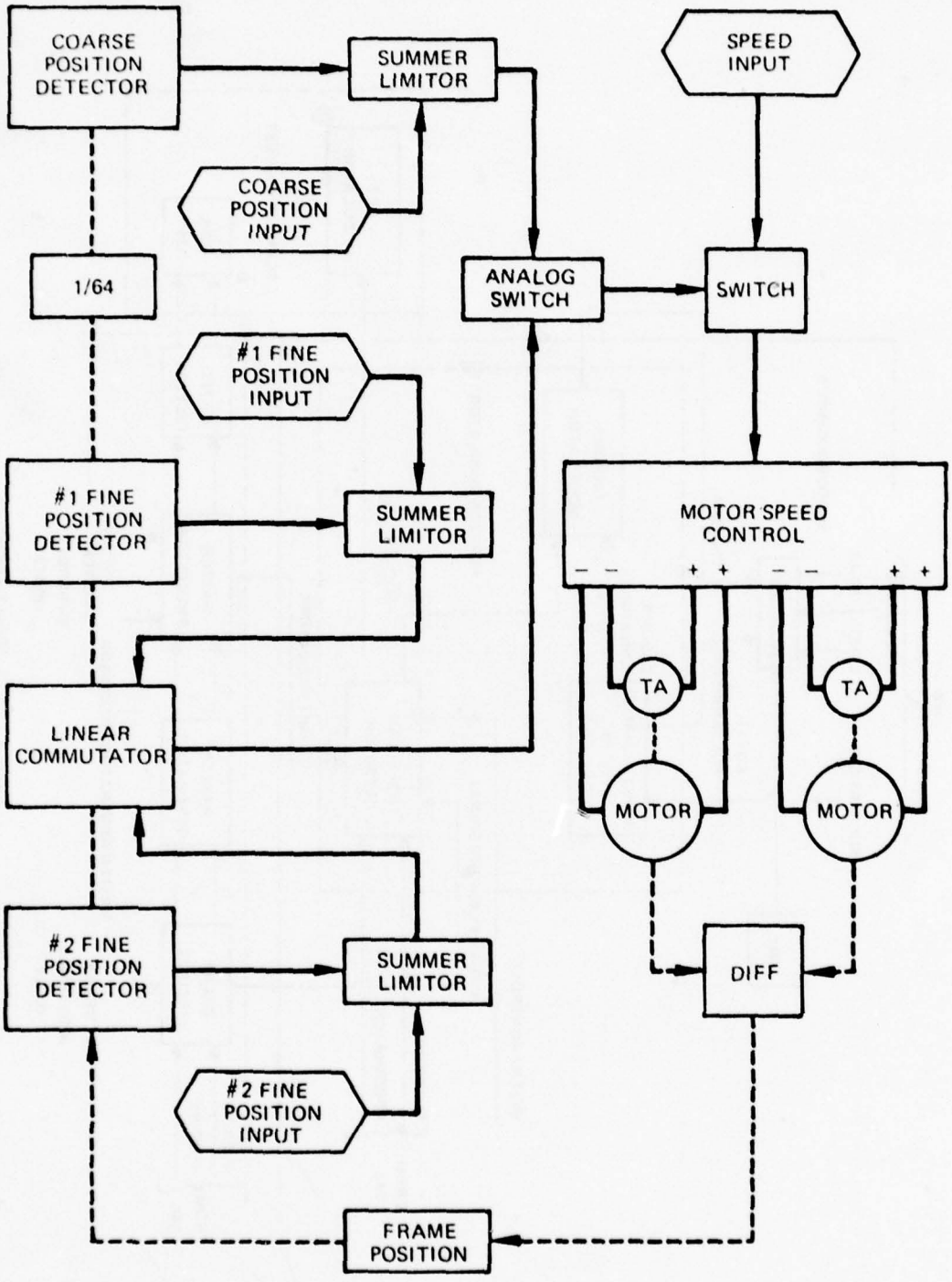


Figure 2.1.3.2-12. SERVO system.

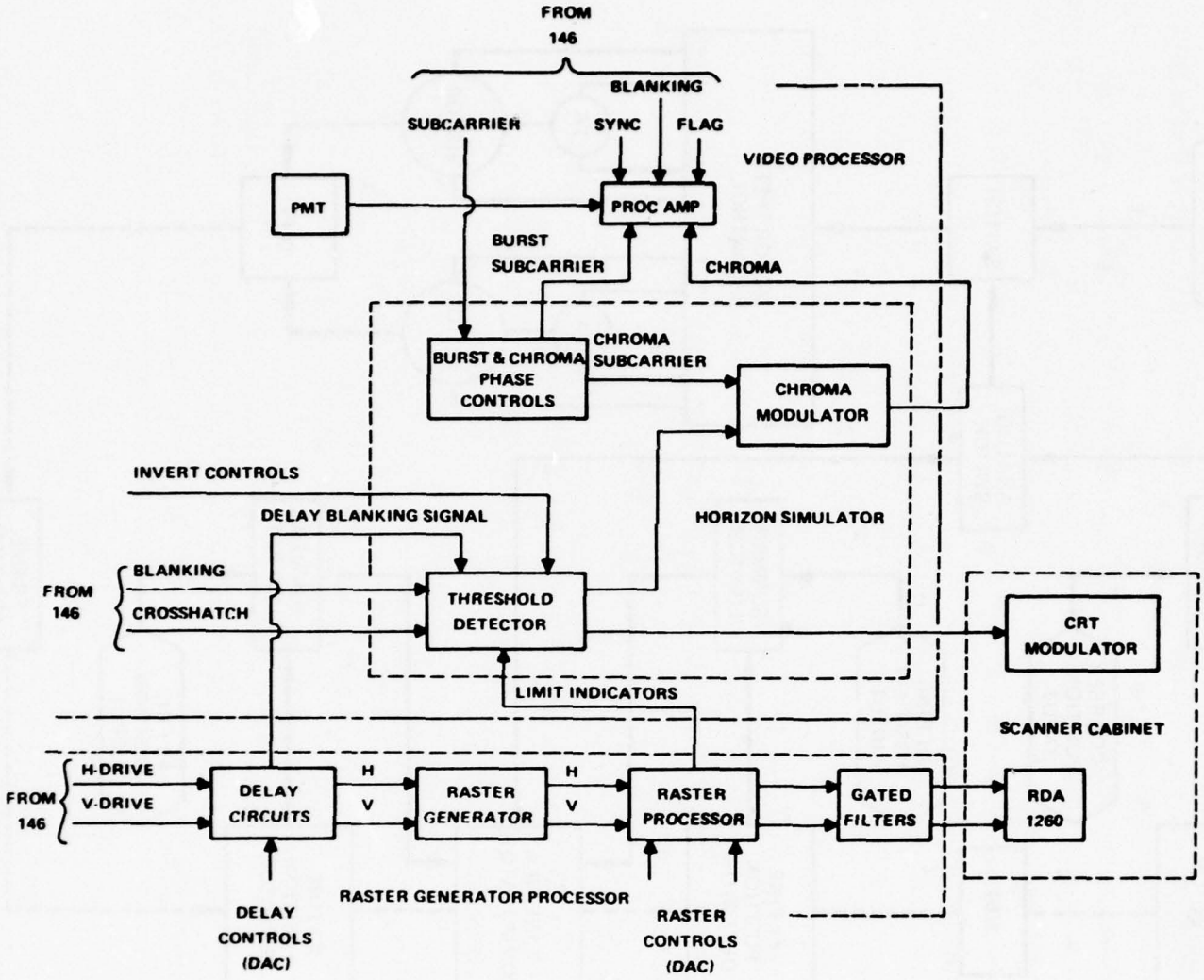


Figure 2.1.3.2-13. Video Processor.

The Horizon Simulator consists of four parts—BURST and CHROMA phase controller, threshold detector, CHROMA modulator, and CRT modulator.

The phase controller produces 2 3.58 MHz signals with independent phase controls. The BURST controls the phase of the BURST on the video output relative to master phase from the 146. In this manner, compensation can be made for the phase shifts caused by propagation delays. The CHROMA SUBCARRIER phase control determines the HUE of the CHROMA generated by the CHROMA modulator.

The CHROMA modulator receives a signal from the threshold detector and outputs a CHROMA signal controlled in HUE by the phase controller and adjustable in SATURATION and LUMINANCE by separate controls. By using the three controls the inserted sky can be set to any color within NTSC capabilities. The intensity of the inserted CHROMA is also modulated by the threshold detector to produce a smooth transition from ground to sky.

The threshold detector performs several functions. The mathematical equivalent of a sensor looking at the horizon is, for some terms, going to infinity. The scan signals from the Raster Processor are level detected to determine the occurrence of the infinity terms. Upon detecting an infinity condition, the threshold detector outputs two signals. One signal drives the CHROMA modulator and "turns on" the sky while the other signal simultaneously blanks the CRT and eliminates the video information from the PMT's. The output from the threshold detector is not a binary signal but is a fast ramp designed to give a "soft" appearance to the horizon. The "threshold.' and "horizon gain" controls adjust the detection point and output signal slope respectively.

Since the threshold detector outputs a signal capable of blanking the CRT, the composite BLANKING signal from the 146 is supplied to the circuitry to blank the raster during retrace. Similarly, a crosshatch signal can be applied to the raster for test purposes.

The raw video at the output of the PMT contains no synchronization information. A monitor could not operate on the PMT output alone. The sync signals added by the Proc Amp provide the necessary timing information. The raster which produces the light input to the PMT is timed by only the H and V DRIVE pulses supplied to the raster generator circuitry. Since the SYNC added by the Proc Amp and the H and V DRIVE used by the raster generator are from the same source, the PMT video will be properly timed, relative to the added SYNC. If, however, the H and V DRIVE signals are phase shifted relative to SYNC, the raster will be late in starting a scan. Since a display monitor begins a scan

according to the SYNC, a time lag will be created between the raster and the display. This "time" lag will cause a "position" shift to occur in the video information as displayed on the monitor. Thus, delaying the H and V DRIVE has the effect of moving the picture down or to the side on the display.

As far as the simulation is concerned, the moving of the picture down is equivalent to pitching the aircraft up toward the sky. The raster delay circuitry performs this task, and the delays are continuously variable and may be set by computer control. The delay circuitry also generates a signal during the period between the input drive pulse adn the delayed output pulse to drive the horizon simulator and turn on the sky. In this manner the picture "wrap-around" is eliminated. A control line on the horizon simulator inverts the signal from the delay circuits, resulting in a sense of moving the picture up, rather than down. This discussion made reference only to vertical delays and movements; however, similar circuitry exists for the horizontal channel and results in left or right movements. The manual and digital-to-analog converter controls are channel independent.

2.1.3.2.2.7 PDP-11 Interface. The PDP-11 interface driving the Video System and the Flying Spot Scanner is based on a circuit designed for the DAIS Hot Bench. The interface built for the Video Center is for data output only with no provisions for input data from the Video Center to the computer. Furthermore, the Video Center accepts digital data, hence no synchro or resolver conversion is required.

2.1.3.2.2.8 Interface Software. The purpose of the interface control software is to initiate block transfer of data from the computer memory by providing the necessary commands to the DR11-B. The block transfer is initiated when the interface control software loads the word count register with the 2's complement of the number of words to be transferred, specifies the initial address of the memory block where the transfer is to be made, selects the interface to receive the block transfer, appropriately setting bits 2 and 3 (function 2 and function 3) of the command/status register, and issues a GO pulse by loading bit 0 of the command/status register.

As the transfer is taking place, the control software monitors bit 7 (the ready bit) of the command/status register. Detection of a 1 in the ready bit indicates the completion of the block transfer.

2.1.3.2.2.9 Scanner Video Control. The Raster Generator/Processor is controlled by DC voltage inputs. These voltages are supplied from seven DAC cards. Each DAC card contains two sections of eight bits each (two BCD digits) and a polarity controller. The

output voltage range is from -9.999 volts to +9.999 volts. To transfer data, a buffer is set up with the 14 words. The lower eight bits of a word must form the bit pattern corresponding to the two BCD digits to be transferred. The ninth bit of a word is the sign bit with "0"=+ and "1"=-.

Table 2.1.3.2-3 indicates the control function of each DAC.

The following is a description of the control functions with appropriate limits:

1. The size controller corresponds to slant range or altitude (limits +1500 to +9999);
2. Heading is the compass heading with +5000 equal to North. (limits -9999 to +9999);
3. Roll changes the perspective look angle about the roll axis; +5000 is straight down; below 5000 is left and above 5000 is right (limits +500 to +9000);
4. FOV or field-of-view simulates various sensor fields-of-view and is limited by approximation to $\pm 25^{\circ}$; exact calibration is subjective, but a good guess is that 25° is +4000; therefore, the limits are +0 to +4000.
5. Forward depression or pitch is similar to roll except about the pitch axis; +5000 is straight down with +400 to +5000 corresponding to a forward look angle (limits +400 to +9200);
6. The delay control DAC's are wired to output only positive voltages; the sign bit has been wired to the "invert" line of the "raster delay" circuits.

2.1.3.2.2.10 Switcher Control. The Routing Switcher may be controlled by the PDP-11 through the interface. The lower six bits of the lower bits are the binary number of the y address. The lower four bits of the upper bits is the x address. By creating a word in memory with the appropriate bits set, the switcher can be commanded from the computer.

TABLE 2.1.3.2-3. DAC CONTROL FUNCTIONS

Data Word	Data Section	Output	Data Word	Data Section	Output
1	1A		8	4B	FOV
2	1B	Size	9	5A	
3	2A		10	5B	Pitch
4	2B	Heading	11	6A	
5	3A		12	6B	Vertical delay
6	3B	Roll	13	7A	
7	4A		14	7B	Horizontal delay

2.1.3.2.3 Summary

The purpose of the Video Control Center is the control and generation of video signals for the simulation laboratory. The routing of video is by solid-state switching and can be controlled by computer. Circuitry for video generation by the Flying Spot Scanner is also under computer control. Steps were taken to provide similar control for the video tape system. All systems are also provided with manual controls. The capability also exists for video from film.

Monitors are available to view all video signals passing through the system. Additionally, test signals and monitoring equipment are available to evaluate the quality of the video signals.

2.1.3.3 Cockpit

2.1.3.3.1 Introduction

A two place, side-by-side cockpit simulator of F-111 dimensions was fabricated as a multipurpose test bed to evaluate controls and displays integrated with the overall avionics systems as shown in Figure 2.1.3.3-1. The complete computer simulation includes the cockpit with controls and displays wired to the other elements of the system in terms of hardware or simulation models, with the cockpit rigged for realistic flight simulation. Man-machine interfaces are also evaluated in the realistic flight environment, using outside visual cues related to the missions. These efforts are centered around the engineering problems involved in integrating the sensors, processors, and displays/controls in the digital aircraft, and the human factor problems involved in piloting this aircraft. Another issue to be considered is redundancy and failure analysis.

Control and display equipment can be arranged to include a Head-up Display (HUD), Horizontal Situation Display (HSD), Vertical Situation Display (VSD), Multi-purpose Displays (MPD's), Control Panels, Multi-function Keyboards (MFK's), etc. The total cockpit configuration can be rapidly and easily changed to suit a particular equipment functional evaluation.

2.1.3.3.2 Simulator facilities

The cockpit simulator functions in an interconnection of AVSAIL facilities as shown in Figure 2.1.3.3-2. A brief functional description of each system element is provided in the following sections.

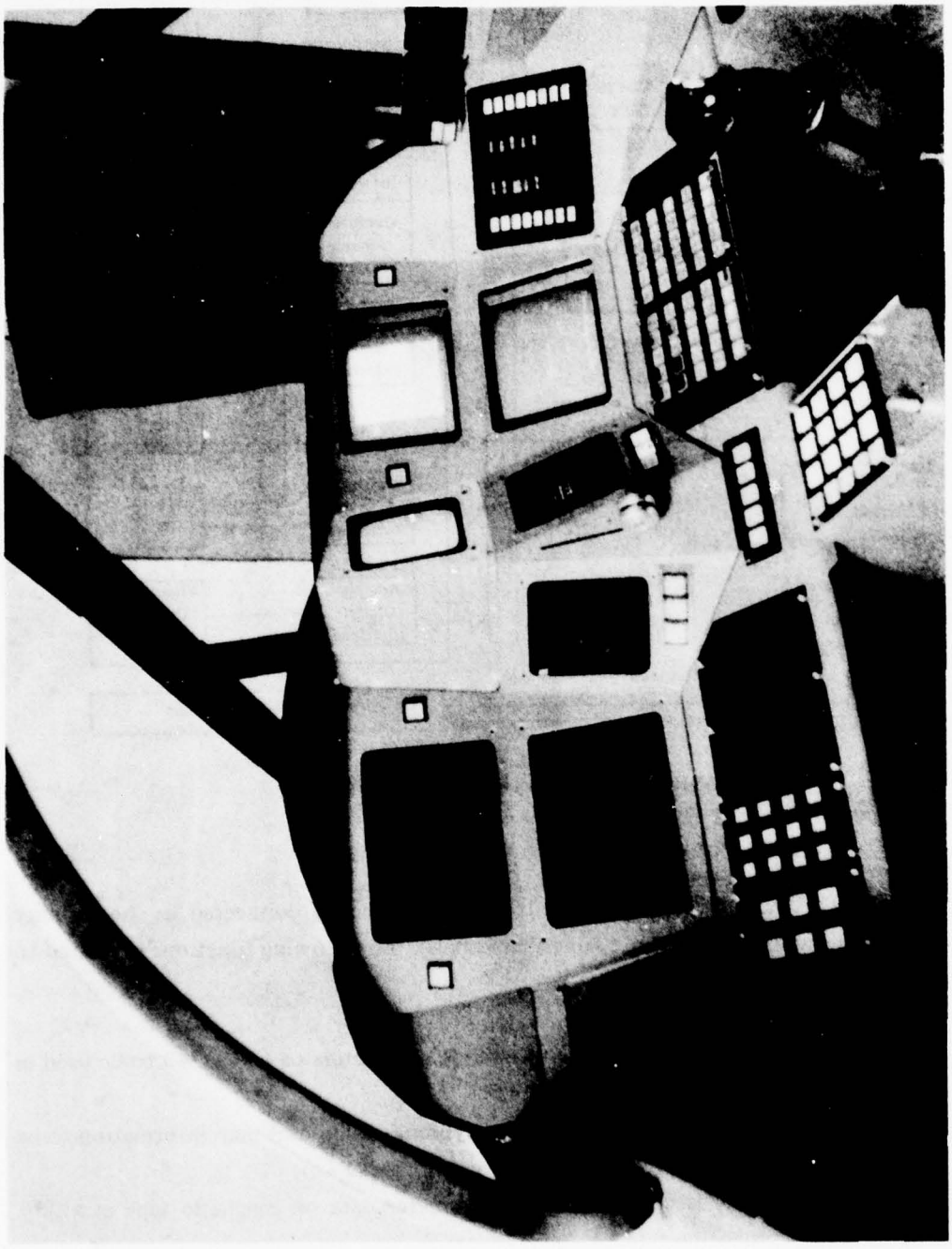


Figure 2.1.3.3-1. The cockpit simulator.

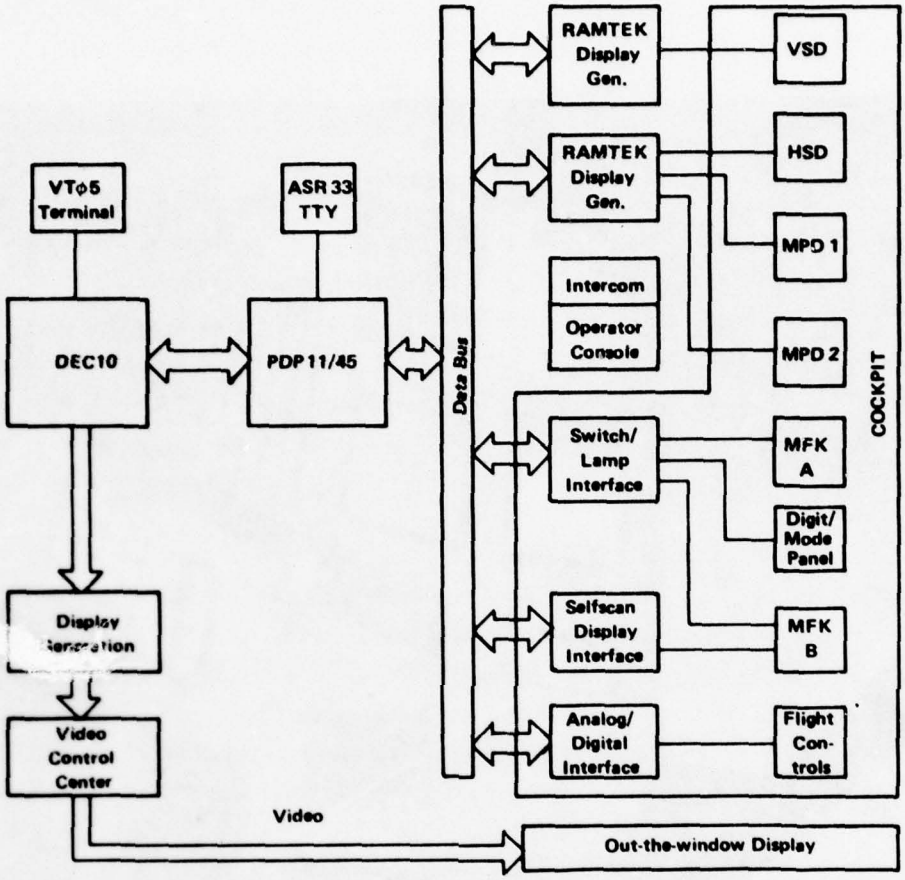


Figure 2.1.3.3-2. System hardware block diagram.

2.1.3.3.2.1 DECsystem 10. The DEC-10 host processor is connected as the primary system controller and provides, on a time-shared basis, the following functions as related to operation of the cockpit:

Simulation Models: — provides all necessary aircraft parameters to the 11/45 to be used in display processing;

Map Assembly: — generates a display list of symbolic waypoint map information to be processed by the Ramtek symbol generator;

Data Recording: — records cockpit display parameter data on magnetic tape at a 20/s iteration rate;

Data Reduction: — an off-line program reduces the raw real-time recorded data into meaningful data that can be analyzed.

2.1.3.3.2.2 PDP 11/45 Computer Interface. This general purpose digital computer provides the following functions:

1. Configuration Control — used to set up the cockpit controls/displays configuration prior to each flight;
2. Display Assembly — generates image listings to be further processed by the Ramtek raster symbol generator; data from the simulation models was used for the VSD and MPD formats.
3. Map Driver — provides output control of map data to the Ramtek symbol generator; the image lists of the map are done by the DEC 10;
4. Keyboard Logic — processes incoming switch data and determines the display state of all the keyboards;
5. Flight Control Sampling and Scaling — buffers and scales flight control data to be used by the DEC-10 simulation models.

2.1.3.3.2.3 Out-The-Window Display. In order to present a more realistic environment to the pilot when the cockpit simulator is being used for man-in-the-loop experimentation, a large-scale parabolic projector screen is mounted forward of the cockpit. The display information is projected onto the screen by video projectors driven from the Video Control Center. This display presents a moving real-time scene which changes in response to the pilot's control actions to the cockpit controls.

2.1.3.3.2.4 Ramtek Display Generators. These generators drive 525 line raster monitors in the cockpit in response to processed image lists and provide the vertical and horizontal situation displays described subsequently.

2.1.3.3.2.5 Cockpit Functional Hardware. The cockpit layout as shown in Figure 2.1.3.3-1 is arranged with the pilot seated in the right side of the cockpit while the left seat is occupied by an experimenter. The controls and displays for the left seat may or may not be activated. As the cockpit is considered a general purpose test facility, a large variety of test configurations are possible. The basic elements of the cockpit functional capability are described below.

Provided as part of the cockpit is a Keyboard Input/Output interface which stores a switch image buffer of all cockpit switch states to be sampled by the 11/45. Also decoded are keyboard lamp data which are sent from the 11/45 to the cockpit to selectively turn lights on or off. The keyboard input/output interface provides the capability of sampling and storing the states of 255 switches and driving 255 lamps. A similar Flight Control

function is provided which digitizes analog stick, rudder, and thrust control inputs and buffers the resultant data for transmission to the 11/45 computer.

Four electro-optical displays are available to provide information for utilization by the Vertical Situation Display (VSD). Figure 2.1.3.3-3 depicts a typical VSD format. A nine-in. diagonal color display presenting the following symbology consisting of: (1) an orange horizon line delineating the boundary between a blue sky and brown earth background, (2) a white pitch ladder with 5 and 10 degree pitch lines, (3) black roll indexes every ten degrees (± 60 degrees) with a white roll index marker, (4) a flight director symbol in orange, and (5) a fixed black aircraft symbol. Altitude, indicated airspeed, heading, vertical velocity, and acceleration (g's) are presented digitally in white on black background. In addition, trend information for the airspeed and altitude parameters is provided by white thermometer type bars which are placed above the respective digital readouts.

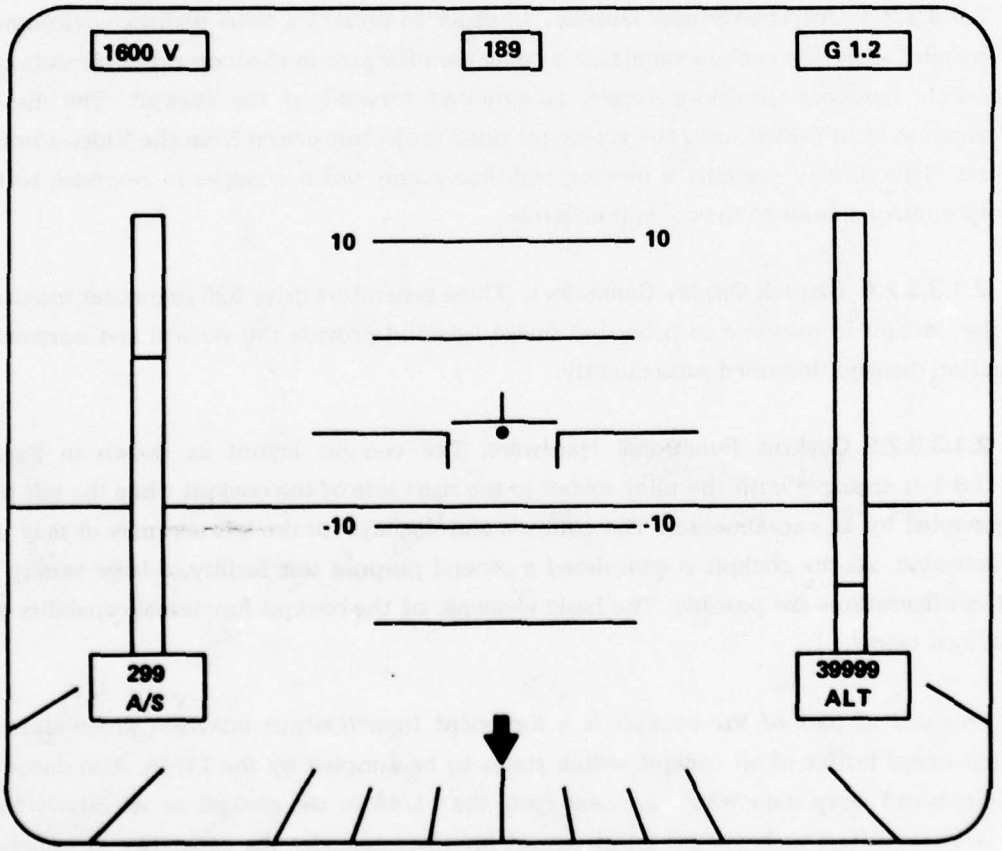


Figure 2.1.3.3-3. Typical Vertical Situation Display (VSD) format.

A nine-in. diagonal color monitor presents simplified navigation information for the Horizontal Situation Display (HSD) in the form of an electronic map in a heading-up or north-up format. The symbology consists of: (1) a triangular aircraft symbol, (2) a symbolic flight path between mission waypoints, and (3) digital readouts of ground speed heading, and distance to the next waypoint. All symbology is green on a black background. Figure 2.1.3.3-4 depicts a typical Horizontal Situation Display format.

Two five-in. diagonal monochrome monitors are used for Multipurpose Displays (MPD's). The one mounted on the left side of the cockpit provides either communication

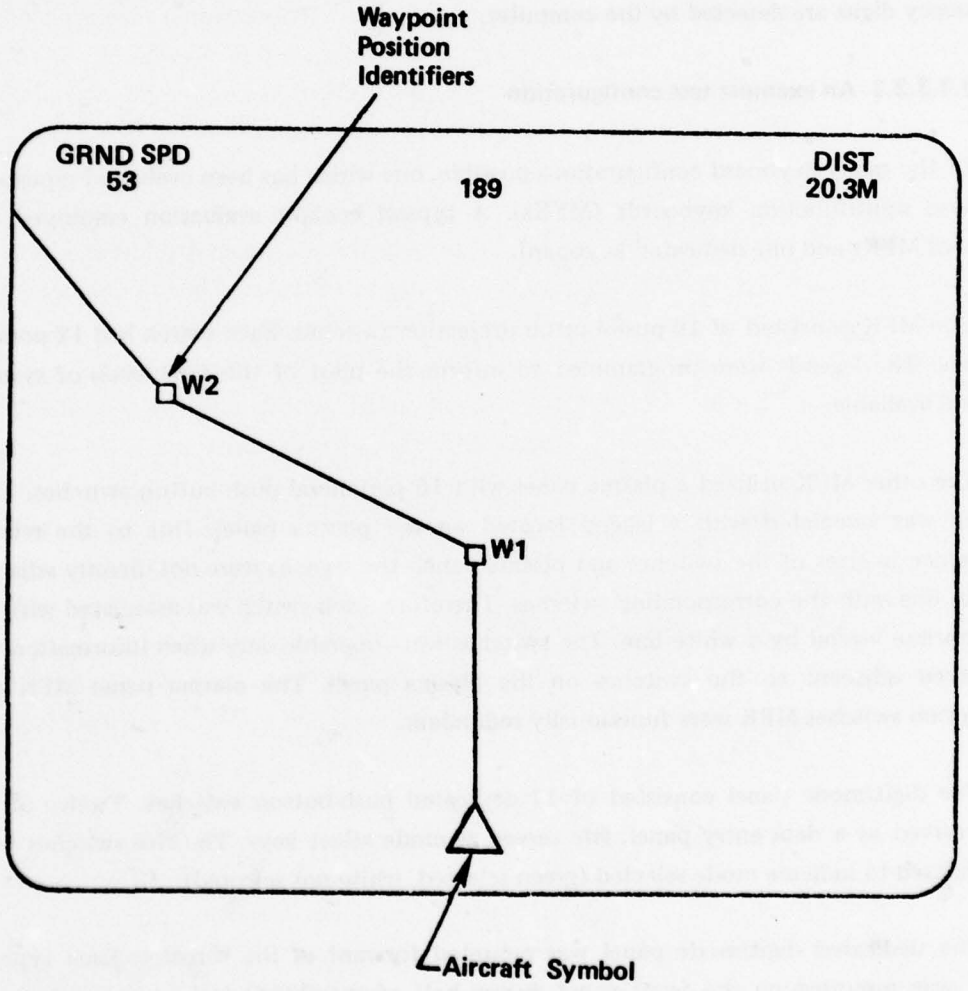


Figure 2.1.3.3-4. Typical Horizontal Situation Display (HSD) format.

data or navigation data. Engine instrumentation and keyboard failure status information are displayed on the right MPD. This MPD has sixteen peripheral push button switches which are available for general use. Each monochrome monitor's presentation is typically formatted in a conventional text mode, however this is not a requirement.

The console's four, six-in. diagonal, monochrome CRT displays provide the experimenter with an experimental console for monitoring the simulator displays (VSD, HSD, and 2 MPDs). The experimenter is also able to initiate tasks and control the normal/failure status of the configurations. In order to minimize experimenter workload, cockpit reconfiguration is automated as much as possible and incorrect waypoint and frequency digits are detected by the computer.

2.1.3.3.3 An example test configuration

Of the many keyboard configurations possible, one which has been evaluated repeatedly employs multifunction keyboards (MFKs). A typical cockpit evaluation employed two types of MFKs and one dedicated keyboard.

One MFK consisted of 16 push-button projection switches. Each switch had 12 possible legends. The legends were programmed to inform the pilot of the four levels of systems control available.

The other MFK utilized a plasma panel with 16 peripheral push-button switches. Each switch was associated with a legend located on the plasma panel. Due to the relative difference in sizes of the switches and plasma panel, the legends were not directly adjacent and in line with the corresponding switches. Therefore, each switch was associated with the appropriate legend by a white line. The switches were operable only when information was displayed adjacent to the switches on the plasma panel. The plasma panel MFK and projection switches MFK were functionally redundant.

The digit/mode panel consisted of 17 dedicated push-button switches. Twelve of the keys served as a data entry panel; five served as mode select keys. The five switches were backlighted to indicate mode selected (green-selected, white-not selected).

The dedicated digit/mode panel was mounted forward of the throttle. Each type of MFK was mounted on the front panel during half of the flights and on the right hand console during the other flights. Thus, the operation of each MFK type, both as a primary (front instrument panel) keyboard and as a backup (right console) keyboard, could be examined.

At the initialization of each test flight, the displays were in the following configuration: (a) VSD—Flight parameters were appropriate to that of level flight in a cruise mode with an altitude of 20,000 feet and indicated airspeed of 301 knots; (b) HSD—Aircraft position was approximately 7 miles short of waypoint 1; the heading was the same as that for the first leg. The ground speed was 301 knots; (c) Left MPD—Engine status format was displayed; (d) Right MPD—Engine status format was displayed. The SENSORS mode at logic level 1 had been activated on the appropriate keyboard.

Throughout each flight, the information displayed on the VSD and HSD was dynamic in response to thrust, bank, and pitch inputs. However, the flight director on the VSD was inoperable until the pilot crossed waypoint 1. Selection of COMM or NAV on the appropriate keyboard determined whether communication or navigation status was displayed on the left MPD. When displayed, the navigation status on the left MPD constantly presented new information such as aircraft position, time, distance to the next waypoint, etc. Alsc, the communication format display presented the status of the communication radios.

The appropriate keyboards were inoperative until activated by the experimenter. Activation of the following switches was required for a navigation update: NAV, NAV COMP, WAYPT, and the appropriate digits. If an incorrect switch was pushed in levels 1, 2, or 3, legends unrelated to the requested task appeared on the keyboard. In order to complete the required task, the pilot had to correct the mistake. To accomplish this, the pilot pushed the CLEAR switch once to return the keyboard to the previous level and then made the proper selection. Once the pilot reached the fourth logic level step, a pre-entry readout of each selected digit was available on the navigation MPD format. When the pilot pushed the ENTER button, the computer interpreted the digits selected and determined their accuracy. If the pilot had pushed an incorrect digit, the error message, "INCORRECT DIGIT ENTRY, RE-ENTER," was presented at the top of the display and the pilot would re-enter the correct data. Once the data was correctly entered, the task would be complete; the keyboard would reinitialize to logic level one and then become deactivated.

For a communication change, activation of the following switches was required: COMM, UHF and UHF POWER, A/N and appropriate digits. Both UHF and UHF POWER switches were selected at step 2. However, during the second communication change of each flight, the UHF power remained active so that the required switch sequence became: COMM, UHF, A/N, and its digits. The MPD/keyboard changes and related procedures of a communication change were similar to that of a navigation update. However, if a pilot entered an incorrect frequency that was still within the normal UHF

frequency range, the keyboard returned to the third level, but no error message was presented on the MPD. The pilot was then notified by the experimenter to redo the task.

Concerning the normal/failure status of the configuration, each flight was initialized in a normal mode. When the experimenter changed the configuration to a degraded mode, the master caution light, located to the left of the VSD, flashed orange. When the pilot acknowledged the failed state by pushing the master caution light, the flashing stopped. Concurrently, a primary keyboard became inoperable and the logic levels that were on the keyboard were presented on the right console keyboard. When a failure occurred, the display on the right MPD was replaced with the failure message. This format specified which keyboard was failed and the logic levels that were operable on the right console, backup keyboard. When the experimenter returned the cockpit to a normal mode, the master caution light flashed green, the primary keyboard became operable, the right console keyboard became inoperable, and the information displayed on the right MPD indicated that normal operation was reinstated.

Other controls used included: (a) flight mode select panel—only the cruise mode was utilized in this study; (b) landing gear control panel—speed brakes and flaps were operational while landing gear was not; and (c) pitch indication zeroing switch—activation of this blue-lighted push-button switch aligned the horizon line with the aircraft symbol at that pitch altitude.

Thrust was controlled by a left-side, slide control throttle. Bank and pitch commands were input either by means of a center stick mounted on the floor or a side stick mounted on the right console. This latter configuration also had a right side armrest.

2.1.3.3.4 Future development

Proposed additions to the cockpit facility include a Head-up Display and a Helmet Mounted Display. The intent of these displays will be to present well concentrated flight data to the pilot in order that it will no longer be necessary to observe continuously a number of display panels. Instead, the pilot can concentrate on one or perhaps two displays mounted directly in front of him and thereby be able to fly at all times in a "head-up" mode.

SECTION 2.1 BIBLIOGRAPHY

2.1.1 and 2.1.2 DEC-10 Hardware and Software

AFAL. DECsystem-10 Newsletter Number 1.

AFAL. DECsystem-10 Newsletter Number 2.

Borasz, F. M., and Burlakoff, Mike. "JOVIAL (J73/1) Language Specification." AFAL Report SA204200. December 19, 1974.

DECsystem-10 Introductory Manuals. DEC-10-XFIMA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, 1974.

DECsystem-10 Monitor Calls Manual. DEC-10-OMCMA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, 1974.

DECsystem-10 Operating System Command Manual. DEC-10-OSCMA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, 1974.

DECsystem-10 Technical Summary. Digital Equipment Corporation.

DECsystem-10 Text Editor and Corrector Program Programmer's Reference Manual. DEC-10-UTPRA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, April 1975.

DECsystem-10 Utilities Manual. DEC-10-UTILA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, 1975.

DECsystem-10 Utilities Manual Addendum. DEC-10-UTILA-A-DNI, June 1975.

Getting Started with DECsystem-10. DEC-10-XGSDA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, March 1975.

Green, T. "DAIS Simulator (GT-44) C PDP-11/40—Part of: *AVSAIL* Revised Sketch." Unpublished, March 1977.

Green, T. "Evans and Sutherland Picture System with PDP 11/50—Revised Sketch." Unpublished, March 1977.

Green, T. "System Flow Diagram—Video Center, Picture System, Cockpit, PDP-11/45, PDP-11/40, PDP-11/20, DEC-10,—Revised Sketch." Unpublished, March 1977.

Green, T. "DEC-10 System Host Simulation Processor Hardware—Revised Sketch." Unpublished, March 1977.

Green, T. "Systems Integration Lab. (SIL) PDP-11/20—Part of: ***AVSAIL*** Revised Sketch." Unpublished, March 1977.

Green, T. "DAIS Simulator (GT-44) A PDP-11/40—Part of: *AVSAIL* Revised Sketch." Unpublished, March 1977.

Green, T. "DAIS Simulator (GT-44) B PDP-11/40—Part of: *AVSAIL* Revised Sketch." Unpublished, March 1977.

Green, T. "Systems Integration Lab (SIL) PDP-11/45—Part of: *AVSAIL* Revised Sketch." Unpublished, March 1977.

Green, T. "F-16 Simulator Configuration." Unpublished Sketch, March 1977.

Hara, Jeff, and Lynn, W. *JOVIAL J73/1 Computer Programming Manual*. AFAL Report MA204200-1. January 5, 1976.

Hara, Jeff, and Lynn, W. *JOVIAL J73/1 DEC-10 Programming Languages Interface Manual*. AFAL Report MA204200-2. January 5, 1976.

Hara, Jeff, and Lynn, W. *JOVIAL J73/1 Language Reference Handbook*. AFAL Report MA204200-3. October 1975.

Hoskins, P. "DAIS Simulator (GT-44) C PDP-11/40—Part of: *AVSAIL* PLN. NR. 10." AFAL Data Item 75-0002. April 15, 1975.

Hoskins, P. "Systems Integration Lab. (SIL) PDP-11/20-Part of: *AVSAIL* PLN. NR. 22." AFAL Data Item 75-0006. April 15, 1975.

Hoskins, P. "DAIS Simulator (GT-44) A PDP-11/40—Part of: *AVSAIL* PLN. NR. 10." AFAL Data Item 75-0004. April 15, 1975.

Hoskins, P. "DAIS Simulator (GT-44) B PDP-11/40—Part of: *AVSAIL* PLN. NR. 23." AFAL Data Item 75-0007, April 15, 1975.

Hoskins, P. "Systems Integration Lab (SIL) PDP-11/45—Part of: *AVSAIL* PLN. NR. 10." AFAL Data Item 75-0003. April 15, 1975.

Hoskins, P. "Systems Integration Lab (SIL) PDP-11/45—Part of: *AVSAIL* PLN. NR. 23." AFAL Data Item 75-0007, April 15, 1975.

JOVIAL J73/1 Retargetability Guideline Manual. Data Item AOOD. Computer Sciences Corporation, April 1976.

JOVIAL J73/1 Rehostability Procedure Manual. Data Item AOOC. Computer Sciences Corporation, April 1976.

"PDP-11 Computer Family Products and Services." Digital Equipment Corporation, 1976.

"PDP-11 System Report," Digital Equipment Corporation." Report No. 5405.011.100, 1975.

Processor Handbook. Digital Equipment Corporation, 1973.

Sammett, Jean E. *Programming Languages; History and Fundamentals*. Prentice-Hall, 1969.

2.1.3.1 The Picture System

Dresel, D. "Figure B-1 PDP 11/50 System to Simulate HUD." Unpublished AFAL Sketch, November 1, 1976.

Evans and Sutherland. *The Picture System Users Manual*. Computer Corporation, No. E5-PS-5001-005, 1976.

Green, T. "System Overview." Unpublished AFAL Sketch, March 1977.

2.1.3.2 Video Control Center

Green, T. "Video Center and Flying Spot Scanner—Revised Sketch." Unpublished February 1977.

Williams, D. A. "Video Generation and Control System." Ohio: Wright-Patterson Air Force Base, Avionics System Simulation Branch. Report No. AFAL-TR-76-164, August 1976.

2.1.3.3 Cockpit

Dresel, D. "Cockpit Simulation Facility." AFAL Internal Publication, January 1977.

Reising, Dr. T. M., and Bateman, R. P. "The Use of Multi-Function Keyboards in Single-Seat Air Force Cockpits." AFAL Technical Memorandum AFFDL-TM-76-67-F6R, July 1, 1976.

2.2 AVIONICS EVALUATION PROGRAMS

The Avionics Evaluation Program (AEP) is a collection of avionic performance assessment models. The AEP provides convenient and systematic assessment of avionics in the mission environment. The program is designed to be flexible and easy to use with emphasis on realistic consideration of the operational environment and the generation of useful data. AEP can be utilized for analysis of most air-to-ground and air-to-air missions. Individual programs contained within AEP include air-to-ground and air-to-air mission analysis, weapon delivery error analysis, target acquisition analysis, and a one-on-one dogfight analysis. These programs are implemented in a conversational interactive mode thus providing a powerful analytical tool. These programs are available to users by means of dial-up terminals and therefore are available throughout DOD as well as contractor organizations.

2.2.1 AEP Program Capability

2.2.1.1 Air-to-Ground Mission Analysis Programs

The air-to-ground mission analysis program evaluates the performance of a flight of up to four aircraft through a specified number of days of operation. The aircraft proceeds along an externally generated nominal trajectory through the mission phases of takeoff, navigation to the search area, search, attack, and return to base. Consideration of ground service requirements is included. Monte Carlo techniques are applied to MTBF (Mean Time

Between Failure) data for the defined avionics throughout the mission to determine which subsystem modes are functioning, resorting to backup modes and mission aborts as required. The model utilizes the best mode still available for each function at the time it is to be performed. Target location uncertainties and navigation system performance parameters are combined to define the actual flight path, relative to the true target location. The sensor ground swatch for the defined search pattern is then compared to the true target location to determine if the target passes through the sensor ground area coverage. Probabilities of detection, target kill, and aircraft survival are sampled to determine which mission phases are successfully completed.

The following steps are required to set up a problem:

1. The mission must be defined in terms of the target, threat, and weapons;
2. The flight profile is defined using waypoints to describe flight segments;
3. A suite of hardware (black boxes) describing the aircraft is itemized; reliability data are provided for each black box;
4. The required mission functions are selected (typical functions include navigation, navigation update, target acquisition, weapon delivery, survivability, communications, refueling, etc.);
5. For each function, the primary and backup modes of operation must be defined; a mode is defined by specifying the performance capability of the mode (if applicable) and assigning the hardware black boxes required to operate in that mode.

Figure 2.2.1.1-1 shows a typical program output. Another page is printed showing the hardware items and the number of missions on which each box failed or was disabled by enemy fire. Another page shows each function, its associated modes, and the number of times each mode was utilized. The output also includes additional statistical data on the occurrence of ground and airborne events as well as mission cost data.

2.2.1.2 Weapon Delivery Error Analysis Program

The weapon delivery analysis routine is a program for determining the distribution of impact errors for a weapon system utilizing unguided, unpowered bombs. The routine is capable of accommodating almost any weapon delivery mechanization under the following general assumptions:

```
AEPOUT COMMAND
--        EVENTS,ALL

------------------GROUND EVENTS------------------
GENERAL MAINTENANCE                          19
SORTIE CANCELLED                              3
NOT OPERATIONALLY READY                       5
EQUIPMENT ITEM REPLACED                      89
REPLACEMENT UNAVAILABLE                      15

-----------------AIRBORNE EVENTS-----------------
DETECTED FAILURE                            129
FALSE FAILURE                                 9
UNDETECTED FAILURE                           10
AIRCRAFT ABORT
          A/C  1                              0
          A/C  2                              1
AIRCRAFT LOST
          A/C  1                             13
          A/C  2                             25
A/C LOST TO ENEMY FIRE
          A/C  1                              5
          A/C  2                              4
MISSION ABORT                                 5
LOW FUEL ABORT                                0
DISPLAY TARGET DETECTION
          A/C  1                             54
          A/C  2                             62
VISUAL TARGET DETECTION
          A/C  1                             73
          A/C  2                             59
TARGET ATTACKED
        TARGET 1                            127
        TARGET 2                            121
PRIMARY DESTRUCTION
        TARGET 1                             10
        TARGET 2                             10
SECONDARY DESTRUCTION
        TARGET 1                             49
        TARGET 2                             39
GO-AROUND FOR ATTACK
        TARGET 1                              1
        TARGET 2                              5
```

Figure 2.2.1.1-1. Sample output for the AEP air-to-ground mission analysis program.

1. Flat, nonrotating earth,
2. Linear transformation of component error sources to impact errors, and
3. Normal distribution for all error sources.

The routine can be operated in three modes:

1. The user can supply a weapon release algorithm in the form of a FORTRAN subroutine;
2. The user can supply equations which transform the sensor measurements into rectangular coordinate estimates of position, velocity, and wind relative to the target; thirteen system implementations using this technique are presently stored in the program; and
3. The third mode requires a user-supplied FORTRAN subroutine duplicating the weapon release algorithm as in Model 1; this mode applies to systems which employ filtering and for which the user chooses to specify the sensor errors in terms of magnitude and frequency characteristics.

To set up a problem, the user (1) selects or defines an aircraft; (2) describes the trajectory in terms of dive angle, velocity, release altitude, pull-up acceleration, etc.; (3) selects 1 of the 26 stored weapons; and (4) selects contributing error sources (from approximately 50 stored sources). Figure 2.2.1.2-1 shows a typical output.

2.2.1.3 Target Acquisition Analysis Program

The AEP target acquisition model is a modified version of MARSAM II. The Multiple Airborne Reconnaissance Sensor Assessment Model (MARSAM) is a target acquisition program developed in 1968 for the Aeronautical Systems Division, Wright-Patterson AFB, Ohio, by Honeywell, Incorporated. It was originally developed for use in the performance assessment of reconnaissance sensor systems of varied types operating on prescribed flight profiles against ground targets in specified background and weather environments. MARSAM II addresses those aspects of sensor performance related to the capability of such systems to provide target identification detail. Specifically, the types of aerial sensor systems considered in the MARSAM II model are: frame and panoramic cameras, television, the visual observer, vertical and forward-looking infrared, side-looking and forward-looking radar. An active TV model has been added. The FLIR model has been added to include the modulation transfer function (MTF) data as a system performance measure. In addition, there is a stored library of characteristic data for numerous target-elements, background, weather, and terrain conditions.

Weapon Delivery System Number 3
Constant Flight Path Angle Trajectory With an 0.387 Thrust Setting

	Time	TAS	FPA	Altitude	Range	Alpha
Roll-in	−1.000	300.0	−.0	1000.0	4451.6	3.7
Designate	−0.000	300.0	−.0	1000.0	3945.0	3.7
Release	−0.000	300.0	−.0	1000.0	3945.0	3.7
Minimum Altitude	0.020	300.0	−.0	1000.0	3934.8	18.2

	Error Source (units)	Along Track: Error Source Magnitude	Along Track: Miss Distance	Cross Track: Error Source Magnitude	Cross Track: Miss Distance
16	Video Tracker Los Angle (Mils)	3.0	3.0	3.0	12.2
17	Video Tracker Angle Rate (Mils/s)	0.5	34.2	0.4	12.9
18	Laser Range (Ft)	25.0	0.4	0.0	0.0
19	Laser Range Rate (FPS)	5.0	−18.7	0.0	0.0
22	Wind Correction (FPS)	12.5	−3.1	12.5	2.1
25	Laser Alignment (Mils)	2.0	−0.6	0.0	0.0
34	Sideslip (Mils)	0.0	0.0	1.4	5.5
35	Steering (Mils)	0.0	0.0	5.0	19.7
43	Bomb Fall Algorithm (Mils)	1.0	8.9	0.0	0.0
	Prerelease Root Sum Square		40.2		27.1
49	Release Delay (MSEC)	20.0	10.1	0.0	0.0
50	Ejection Velocity (FPS)	2.0	29.2	0.0	0.0
51	Ballistic Dispersion (Mils)	3.0	26.7	3.0	12.2
	Root Sum Square		57.3		29.7
	Probable Errors REP = 38.64		DEP = 20.06		CEP = 50.60

Figure 2.2.1.2-1. Sample weapon delivery output.

MARSAM II models the sensor system and the operational environment in detail. It contains models for displays, lenses, filters, and film. It considers the impact of image motion compensation, platform stabilization errors, backscattering, and atmospheric effects on sensor performance. The human observer is modeled in terms of ability to perceive the target as a function of size and contrast, display signal-to-noise ratio, presence of confusing objects, and time in the field of view. A search performance model in human factors display has been added as an option. Available outputs from MARSAM include detailed sensor system performance parameters and associated probability measures of detection, recognition, and identification.

2.2.1.4 Air-to-Air Mission Analysis Program

The Air-to-Air AEP mission analysis program is a Monte Carlo simulation of two

opposing aircraft flights (up to four aircraft in a flight) through an entire mission. As the flight progresses, it is influenced by hardware failures, refueling, communications to airborne or ground controllers, enemy aircraft detection capability, identification requirements and weapon capabilities. Modeling of these functions varies considerably in levels of detail based upon impact on mission success and initial model development priorities. Detailed visual, radar, and infrared detection models are available for the target detection function.

When one side detects the other, that flight pursues a course directly at the other flight and fires when the weapon constraints are satisfied. The encounter is considered only until both sides have detected the other. At that time, the relative positions and headings are stored for output so that users can determine which side has the relative advantages. Kills occur only if weapons are fired before detection by both sides has occurred. At the termination of the engagement, both sides return to the nominal flight profile for the return flight.

The mission is described by the vehicle hardware makeup, flight profile and mission functions. However, these must be described for both friendly and hostile aircraft. All friendly aircraft (and similarly all hostile aircraft) must be identically equipped, although friendly need not be the same as hostile.

2.2.1.5 One-on-One Dogfight Analysis Program

A separate, deterministic air-to-air program permits analysis of the dogfight encounter. It simulates an engagement between two fighter aircraft. The logic for control of aircraft maneuvers is based on lag pursuit and energy management. Lag pursuit implies that each aircraft attempts to get on the tail of the other. Energy management control implies that the aircraft seeks a velocity and altitude for best turning performance. A comprehensive on-line plot capability supports use of the dogfight analysis program. Users can plot large numbers of position, velocity, and acceleration components in an earth or aircraft reference frame in two or three dimensions. In addition, the user can define firing criteria and the program will automatically examine the results of a run to determine intervals when the criteria are satisfied.

2.2.2 Interactive Graphics Capability

An interactive graphics processor is available for use with AEP. The purpose of this processor is to ease user difficulty in communicating with the computer and with AEP. Specifically, the processor has the following characteristics:

1. The user develops input on-line;
2. Input data is stored on permanent files for later reference or use in other runs;
3. Input data are checked against upper and lower limits to detect improper input;
4. The program is executed in steps to allow the user to check reasonableness of partial executions;
5. Output data can be graphically displayed and hard copies, suitable for presentations or reports, can be made; and
6. A special "help" feature is available to provide the user with instructive comments whenever the user is in doubt about how to proceed.

The AEP programs are batch programs, even though they are automatically executed from a remote interactive terminal. The input and output processors are used to communicate with the user for problem setup and review of the output. New data is stored in convenient subsets for later use. A very important "help" file and a complete on-line user's manual are available to aid the user in communicating with the processor and associated programs.

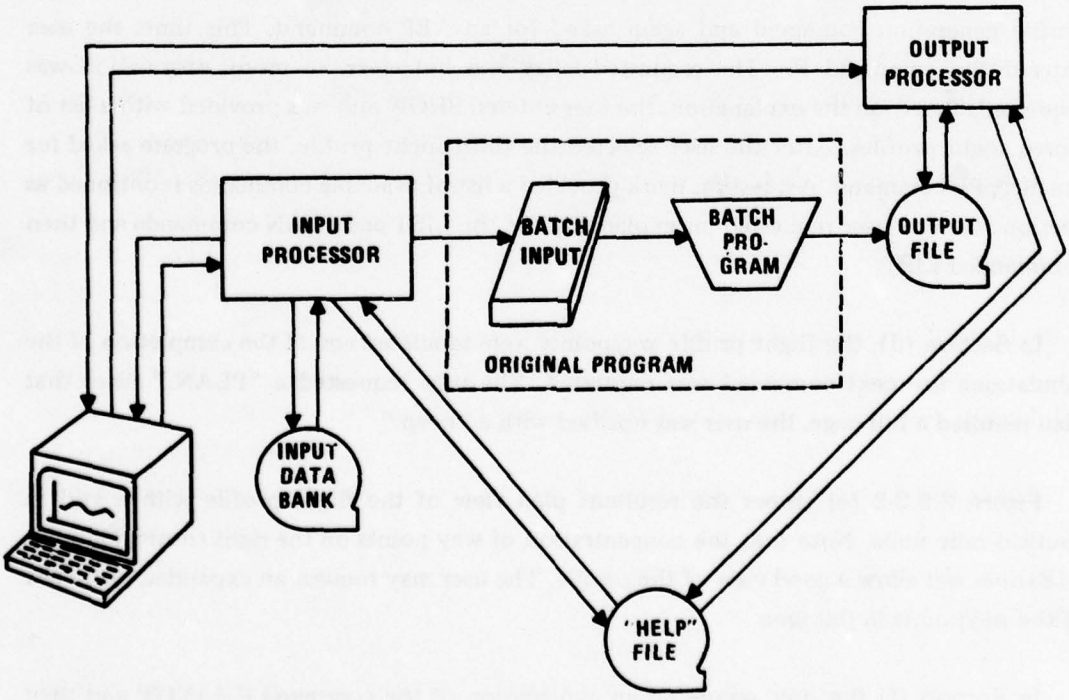


Figure 2.2.2-1. Application of interactive graphics to existing batch programs.

Figure 2.2.2-1 shows an overall block diagram of the interactive graphics programs. The user specifies the required data by either supplying new data or retrieving data previously stored. Once the user is assured that the data are satisfactory, an execution deck can be requested and the program can be executed. The main programs produce an output on a permanent file. The interactive software then interrogates the output permanent file at the user's command to produce the desired displays. The sample air-to-ground mission analysis problem of Figure 2.2.2-2 on the following pages demonstrates the manner in which the interactive software can be used. Each section of Figure 2.2.2-2 is a hard copy of the actual display, except for the figure title.

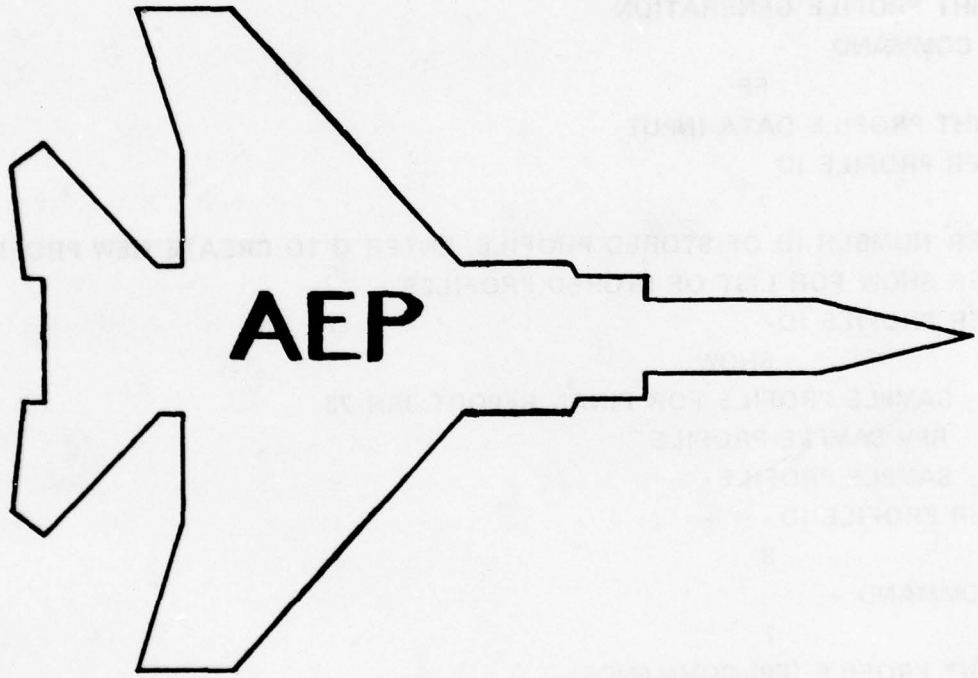
After the user has logged in and attached and loaded the program, the logo of Section (a) appears. Since the logo uses a full page, the user is informed (with an audible "beep" from the terminal) that, at the next step, the page will be erased. Thus, if a hard copy is desired, it should be made at that time. On the next page the user is asked for an AEP command. Note that whenever a double dash (--) appears, the program is expecting a user input. Since the user did not remember the possible AEP commands, a question mark (?) was entered. The program then supplied all of the possible commands and again asked for an AEP command. The user was interested in the command FP but wanted more information about it, so the command FP? was entered. The program responded that FP was the flight profile generation command and again asked for an AEP command. This time, the user entered the command FP. The requested entry was not clear, so more information was requested. Based on the explanation, the user entered SHOW and was provided with a list of stored flight profiles. After the user selected the third flight profile, the program asked for the next FP command. A question mark provided a list of available commands (continued as Section (c). The user requested an explanation of the LIST and PLAN commands and then commanded LIST.

In Section (d), the flight profile waypoints were tabulated and at the completion of the tabulations the next command was requested. The user requested a "PLAN." Since that plan required a full page, the user was notified with a "beep."

Figure 2.2.2-2 (e) shows the resultant plan view of the flight profile with a grid in nautical mile units. Note that the concentration of way points on the right (near way point #18) does not allow a good view of the profile. The user may request an expanded plan view of the waypoints in this area.

In Section (f) the user requested an explanation of the command CHANGE and then changed way point 12. Note that the user changed only the x and y coordinates of that way point.

WELCOME- YOU ARE NOW CONNECTED WITH THE INTERACTIVE
VERSION OF THE AVIONICS EVALUATION PROGRAM



DEVELOPED BY
BATTELLE, COLUMBUS LABORATORIES

(a) LOGO

Figure 2.2.2-2. Sample execution of the AEP interactive program.

```
AEP COMMAND
- -                    ?
EXECUTIVE (AEP) COMMANDS
EQUIP    SUBF    WD    WDDECK    WDOUT    FP
MARSAM    AEPDECK    AEPOUT    END (QUIT)
AEP COMMAND
- -                    FP?
FLIGHT PROFILE GENERATION
AEP COMMAND
- -                    FP
FLIGHT PROFILE DATA INPUT
ENTER PROFILE ID
- -                    ?
ENTER NUMBER ID OF STORED PROFILE, ENTER Ø TO CREATE NEW PROFILE
ENTER SHOW FOR LIST OF STORED PROFILES
ENTER PROFILE ID
- -                    SHOW
  1    SAMPLE PROFILE FOR FINAL REPORT JAN 73
  2    RPV SAMPLE PROFILE
  3    SAMPLE PROFILE
ENTER PROFILE ID
- -                    3
FP COMMAND -
- -                    ?
FLIGHT PROFILE (FP) COMMANDS
      ENTER VALID COMMAND WITH REQUIRED PARAMETER LIST
I    D    O    LIST    FLY    PLAN    ALT    VEL    SAVE    QUIT
FP COMMAND -
- -                    LIST?
```

(b) Interactive Dialogue

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.

```
DISPLAY LIST OF CURRENT VALUES  POINTS ARE RESEQUENCED TO
    REFLECT PREVIOUS COMMANDS
FP COMMAND -
- -                          PLAN?
DRAWS A PLAN VIEW OF THE PROFILE.  ENTER THE COMMAND AS
    FOLLOWS
    PLAN. ID1. ID2
  POINTS ID1 THRU ID2 ARE PLOTTED
  IF ID2 IS OMITTED.  ALL POINTS FROM ID1 ON ARE PLOTTED
  IF BOTH PARAMETERS OMITTED  ALL POINTS ARE PLOTTED
FP COMMAND -
- -                          LIST
```

(c) Interactive Dialogue (continued)

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.

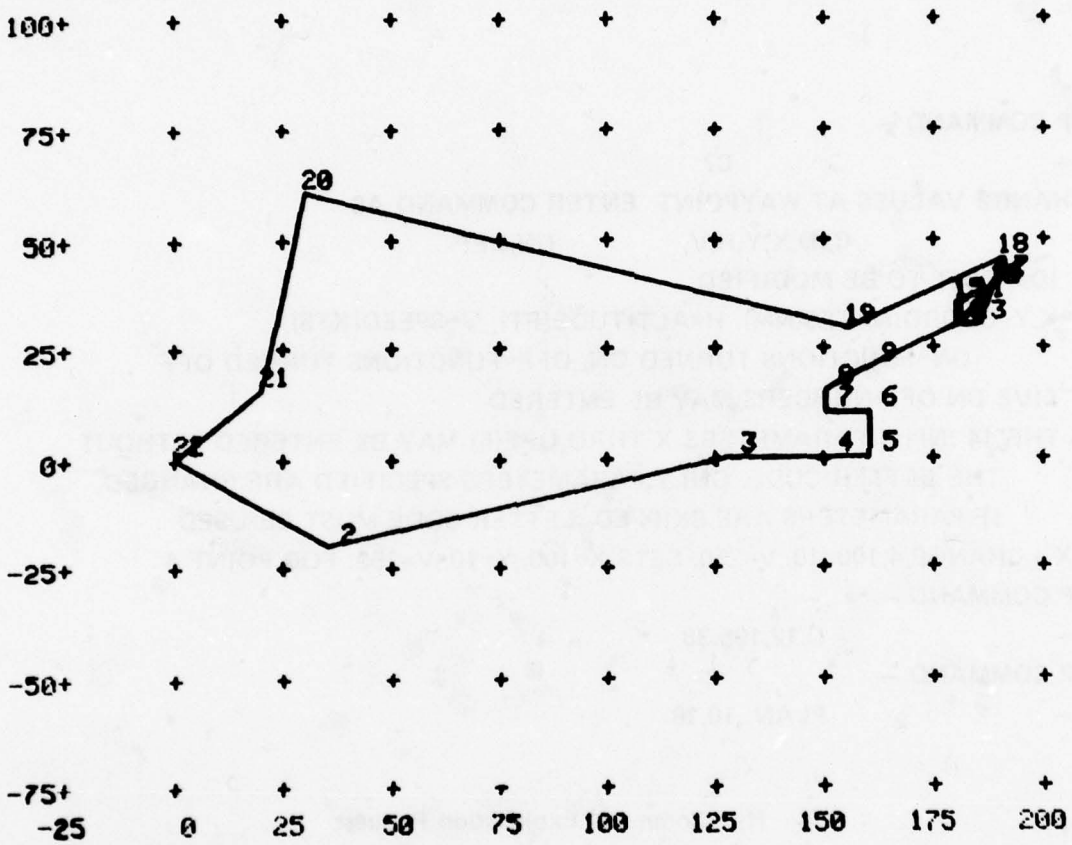
ID	X	X	H	V	ON		OFF	
1	0.00	0.00	0.	150.	701	702		
					1201			
2	35.00	-20.00	20000.	450.				
3	127.50	0.00	20000.	450.				
4	150.00	0.00	5000.	162.				
5	160.00	0.00	5000.	162.				
6	160.00	10.00	5000.	162.				
7	150.00	10.00	5000.	162.				
8	150.00	15.00	5000.	250.				
9	160.00	20.00	5000.	162.				
10	180.00	30.00	5000.	300.	303	901		
11	185.00	35.00	2500.	250.	1102		3	9
12	188.00	38.00	2500.	250.				
13	185.00	30.00	2500.	250.				
14	180.00	30.00	2500.	250.				
15	180.00	35.00	2500.	250.	702		7	
16	189.00	39.00	2500.	250.			11	
17	190.00	40.00	2500.	300.				
18	190.00	45.00	5000.	450.				
19	155.00	29.00	20000.	450.				
20	30.00	60.00	20000.	450.				
21	20.00	15.00	20000.	450.				
22	0.00	0.00	0.	150.			1	12

FP COMMAND --

PLAN

(d) Way Point Description

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.



(e) Flight Profile Plan View

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.

```
FP COMMAND –
––                          C?
CHANGE VALUES AT WAYPOINT  ENTER COMMAND AS
                    C,ID,X,Y,H,V,          ON,OFF
  ID=POINT TO BE MODIFIED
  X,Y=COORDINATES(NM), H=ALTITUDE(FT), V=SPEED(KTS),
       ON=FUNCTIONS TURNED ON, OFF=FUNCTIONS TURNED OFF
  FIVE ON OFF NUMBERS MAY BE ENTERED
  THE 14 INPUT PARAMETERS X THRU OFF(5) MAY BE ENTERED WITHOUT
      THE LETTER CODE. ONLY PARAMETERS SPECIFIED ARE CHANGED.
      IF PARAMETERS ARE SKIPPED, LETTER CODE MUST BE USED
EX– CHANGE,4,100.,10.,V=250. SETS X=100.,Y=10.,V=250. FOR POINT 4
FP COMMAND –
––                    C,12,195.38
FP COMMAND –
––                    PLAN, 10,18
```

(f) Command Explanation Request

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.

```
FP COMMAND –
– –                    SAVE
ENTER TITLE, UP TO 60 CHARACTERS
– –                    SAMPLE FLIGHT PROFILE CREATED 6/25/73
SAMPLE FLIGHT PROFILE CREATED 6/25/73
IS TITLE O.K., Y OR N
– –                    Y
DO YOU WANT TO DELETE ANY TRAJECTORIES, Y OR N
– –                    N
AEP COMMAND
– –                    QUIT
HEP FINISHED
     EXIT
COMMAND–
```

(g) Program Exit

Figure 2.2.2-2 (con.). Sample execution of the AEP interactive program.

In Section (g) the user exited the program. Note that the user must respond to several questions before an exit is allowed. This provides the user with an opportunity to save data and to enter comments for later reference.

This example serves to demonstrate the main features of the interactive graphics processor. It should further be noted that the interactive portions of the program provide the user with a wide variety of displays both on program set-up and during program execution. The various references should be consulted for an indepth discussion of these capabilities.

2.2.3 Program Set-up

In setting up the AEP, missions (i.e., A/G, A/A, etc.) are defined by specifying a flight profile as a sequence of way points with associated tactical functions and by specifying a vehicle hardware makeup as a set of subsystems to support these tactical functions.

2.2.3.1 Flight Profile

Each waypoint along the flight profile will generally reflect a change in direction or airspeed or a change in the functions utilized (the definition of a function will become clear in subsequent paragraphs). Each waypoint is described by providing:

1. the distance east of the origin (base),
2. the distance north of the origin,
3. aircraft altitude above MSL, and
4. aircraft velocity.

From this information, heading and flight path angle are determined by a straight line trajectory between waypoints. Changes in vertical flight path angle occur instantaneously while changes in heading are achieved with a constant bank angle coordinated turn. Changes in velocity occur immediately upon leaving a waypoint. Acceleration incorporates full throttle while deceleration incorporates idle thrust.

2.2.3.2 Functions, Subfunctions, Modes and States

As a part of the flight profile, mission functions are specified over appropriate time intervals. A function is defined as an operation or action performed during the mission. Functions may further be divided into associated sub-functions which are alternative

options for performing a particular function. A table of functions and subfunctions is indicated in table 2.2.3-1 for the air-to-ground and air-to-air versions of AEP. A more detailed description of existing functions and subfunctions is included in subsequent sections.

2.2.3.2.1 Air-to-ground functions and subfunctions

2.2.3.2.1.1 Scheduled Maintenance. The scheduled maintenance function provides an

TABLE 2.2.3-1. A/G AND A/A FUNCTIONS AND SUBFUNCTIONS

Air-To-Ground		Air-To-Air	
Functions	Subfunctions	Functions	Subfunctions
Scheduled maintenance	Preflight	Navigation	Ground controlled intercept
	Thruflight		Airborne controlled intercept
	Postflight	Communications	Interflight communications
Ordinance	General purpose munitions		External communications
	Rockets	Fuel	Fuel utilization
Fueling	Fuel loading		Refueling
	Fuel usage	Target detection	Visual
	Refueling		Radar
Flight	Launch		Infrared
	Inflight aircraft abort	Target identification	Electronic IFF
	Mission abort		Television
	Loss of aircraft		Visual
	Landing	Engagement	Semi-active radar missile
Mission	Schedule		IR missile
	Cost accumulation	Formation	Formation
Formation	Nominal flight	Maneuver	Weaving escort
Navigation	Radio-aided	Weapon detection	Visual
	Self-contained		Radar
Navigation update	Automatic navigation update		Infrared
	Radar navigation update	Mandatory operations	Aircraft abort
	Visual navigation update		Aircraft loss
Communications	Interflight		
	External		
Survivability	Survivability subfunctions (5)		
Target acquisition	Display acquisition		
	Visual acquisition		
Weapon delivery	Manual weapon delivery		
	Automatic weapon delivery		
Target	Target subfunctions (5)		

assessment of the time involved to keep aircraft ready for flight. Three subfunctions are provided to account for preflight (beginning of day), thru-flight (between sorties), and postflight (end of day) maintenance. Data required to describe the ground maintenance are: mean duration, standard deviation, and equipment checklist. The random service time is drawn from a log-normal distribution defined by the input parameters. It is assumed that the repair of items in the checklist occurs in parallel with the maintenance time, and that repair of other items is in series.

The preflight subfunction sequences each aircraft to ordnance loading at the completion of the maintenance interval. If one or more aircraft become NORS, the remainder of the day is cancelled.

The throughflight subfunction sequences each aircraft to ordnance dearming at the completion of maintenance. In reality, dearming would occur prior to ground service. However, for convenience in programming and since the total time interval is the same, throughflight and dearming times are reversed. As with preflight, a NORS state cancels the day.

The postflight subfunction sequences each aircraft to ordnance dearming at the completion of maintenance. Since postflight occurs at the end of the day's sorties, a NORS state has no impact. All redundant items are repaired during postflight maintenance.

2.2.3.2.1.2 Ordnance. The ordnance function calculates the time required to load and arm the ordnance prior to each sortie, and also calculates the time required to unload and dearm the ordnance at the end of each sortie. There are two ordnance subfunctions—general purpose munitions and rockets. The logic within the subfunctions is identical. The difference lies in the distributions which describe the loading and unloading times of the different munitions. The input items for each subfunction are: the number of weapons carried, the mean and standard deviation of the loading, plus arming time per weapon, and the mean and standard deviation of the unloading plus dearming time per weapon.

2.2.3.2.1.3 Fuel. The fuel function provides a means of managing the aircraft fuel requirements. Three subfunctions are available for fuel loading, monitoring of fuel used, and air refueling. The fuel flow rate during flight is one of the aircraft states provided by the aircraft flight simulation. An aircraft abort occurs if fuel monitoring or refueling status are unavailable. A mission abort occurs if no modes are available for air-refueling.

The fuel usage subfunction aborts an aircraft if the remaining fuel is less than that required to complete the flight plus the required reserve.

The air refueling subfunction determines the time of hookup from a uniform probability distribution specified on input by the minimum and maximum hookup times. Several aircraft can be refueled simultaneously, the exact number being specified as input. The time required to refuel is calculated as a function of the amount of fuel to be loaded and the refueling rate, which is also specified as input.

2.2.3.2.1.4 Flight. The flight function provides a means of specifying the equipment requirements for various portions of the mission. Five subfunctions are available: launch, inflight abort, mission abort, aircraft loss, and landing.

The launch subfunction draws a random sortie launch time from a log-normal distribution defined by the input data. This time represents the interval between engine start and takeoff. At takeoff, subfunctions 2-4 are turned on, and launch is turned off. The launch subfunction uses only the time data given with the first mode. A ground abort of the mission occurs if either an aircraft has no available equipment state, or no mode requirement is satisfied.

There are no program calculations associated with the aircraft abort subfunction. The aircraft equipment states associated with this subfunction allow determination of an abort situation. In addition, an aircraft abort will occur if the aircraft does not satisfy some mode requirement. This is a unique use of the mode definition for this subfunction.

The mode/state requirements for the mission abort subfunction are used to define when a sortie must be aborted. If no mode is available, the mission is aborted.

The aircraft loss subfunction is used to define the set of equipment required to keep an aircraft airborne. If no aircraft equipment state is available, the aircraft is lost.

The landing subfunction is used to define the set of equipment required to make a successful landing. If no landing equipment state is available, the aircraft is counted as lost.

2.2.3.2.1.5 Mission. The mission function provides a means of specifying the operations schedule and the cost of various portions of the mission. There are no nominal calculations, aircraft aborts, or mission aborts associated with this function.

The schedule subfunction is the overall mission scheduler for the nominal portion of the simulation. The schedule, utilizing the input data, manages the starting times for the individual sorties and the individual data. All maintenance actions and all flights are

controlled by this subfunction. If a delay occurs in the ground maintenance functions, the scheduler may cancel a sortie if the input maximum delay times are exceeded. This subfunction is called initially to begin the simulation, at the end of throughflight and preflight maintenance, and at the end of each sortie.

2.2.3.2.1.6 Formation. The purpose of the formation function is to specify the position of flight members relative to the lead aircraft. The user specifies the distance right, behind, and above the leader for up to three supporting elements.

2.2.3.2.1.7 Navigation. The navigation function includes two subfunctions—radio-aided navigation and self-contained navigation. The two subfunctions provide the capability of computing and considering navigation errors. Two types of navigation errors are considered—a fixed position navigation error and a per unit time error. A mission abort occurs if the self-contained navigation subfunction fails. If the radio-aided navigation fails, a switch to self-contained navigation is attempted. If the switch is successful, the mission is continued; otherwise, a mission abort occurs.

2.2.3.2.1.8 Navigation Update. The navigation update function includes three subfunctions: automatic update, radar update, and visual update. The navigation update subfunctions calculate the sensor field of view and determine if the checkpoint is within the field of view, and if the checkpoint has been detected. Once the checkpoint has been detected, the accuracy of the navigation update is computed.

2.2.3.2.1.9 Communications. Two communications subfunctions are provided to assess the reliability of the communications equipment: interflight and external communications. Loss of all aircraft equipment states for interflight communications causes an aircraft abort. Loss of all modes for either subfunction causes a mission abort.

2.2.3.2.1.10 Survivability. The survivability subfunctions have two primary purposes. The first is to generate a random time of hit for each aircraft from an exponential distribution specified on input. The second purpose is to process the aircraft hits to assess aircraft damage. The input data items at the constant probability of hit, the per unit time probability of hit, and the probability of aircraft kill. Five subfunctions are provided so that the user may have direct control over when each is used. For example, a typical mission profile may progress through defensive zones with different probabilities of survival. The user can key the use of each subfunction (with different data) to waypoints defining the flight profile.

2.2.3.2.1.11 Target Acquisition. Display and visual target acquisition subfunctions are provided. All targets specified by the user are checked for detection during each search segment, with attack passes occurring based on the sequence of detection. Thus, for relatively close targets, the order of attack can be different than the order of target locations. After an attack against one target, the acquisition process is resumed for the remaining targets, allowing a user to simulate a target of opportunity type mission. There are no aircraft aborts due to loss of acquisition equipment items. A mission abort occurs only if all possible modes of both subfunctions fail.

2.2.3.2.1.12 Weapon Delivery. Manual and automatic weapon delivery subfunctions are available. A significant feature of the updated weapon delivery function is that ordnance and target types are keyed to the delivery modes. This provides the flexibility of considering various ordnance mixes for a multiple target sortie. A unique feature of the weapon delivery function is that a subfunction/mode status is maintained for each aircraft. Thus, it is possible for one aircraft to use the automatic release subfunction and another aircraft to use the manual subfunction. Mode changes due to equipment failure have no immediate impact if weapon delivery has not been activated. When a weapon delivery run is committed and a mode regression occurs, only a mode with a release range matching the committed range will be used. Otherwise, the aircraft with the failure cannot attack on that pass. An individual aircraft does not abort if no equipment states remain. If that aircraft was initially in the automatic subfunction, the manual subfunction will be used if possible. If none of the aircraft can perform the weapon delivery function, the mission is aborted.

2.2.3.2.1.13 Target. The target function manages the number of attack passes and target location uncertainty for up to five targets. This information is specified on input. There are two levels of target destruction accumulated, primary and secondary destruction, which are based on primary and secondary kill radii.

2.2.3.2.2 Air-to-air functions and subfunctions

2.2.3.2.2.1 Navigation Function. The primary purpose of introducing a navigation function is to have a mechanism for reflecting the influence of navigation on target detection. No influence of onboard navigation system errors are considered. Navigation is important on an intercept mission where a supporting ground or airborne element is providing an alert or intercept vectors. The quality of the vectors is not addressed. The effect of the supporting element is to place the airborne flight in an alerted or vectored status. The implications of this status are:

1. In autonomous search (no supporting element), three sweep detections are required to adequately assess target position and direction of flight,
2. In alerted search, two sweep detections are required; and
3. In vectored search, one sweep detection is required.

The navigation subfunctions draw random numbers to determine this status.

The input data items are:

1. Probability that the controller detects the enemy flight, and
2. Probability that sufficient information is available for vectored search.

No action is taken if these subfunctions are lost.

2.2.3.2.2.2 Interflight Communications Subfunction. The interflight communications subfunction calculates a delay in transmitting target detection from one flight member to the rest of the flight. The input data items are:

1. Communication frequency utilization,
2. Mean duration of the utilization,
3. Maximum communication delay, and
4. Time to communicate.

The first input data item is a number between zero and one expressing the probability that the communication frequency is being utilized when a flight member attempts to transmit. The second item is the mean value of the duration of interference. This item defines an exponential probability distribution of delay. The delay is limited by an input maximum delay. This subfunction is called by the detection function. If the user does not select this subfunction, no delay is calculated. If this subfunction is lost, the mission is aborted.

2.2.3.2.2.3 External Communications Subfunction. The external communications subfunction calculates delay in communicating controller alerts or vectors to the airborne flight. The input data items are:

1. Communication frequency utilization,
2. Mean duration of utilization,
3. Maximum time of utilization,

4. Probability of ECM interference,
5. Mean delay of ECM interference, and
6. Maximum vector update period required to maintain vectored status.

The delay is calculated in the same way as for interflight communications, except that jamming of the link is an added possibility. An exponential distribution is used for the duration of jamming when it occurs. The purpose of this subfunction is to determine delay in communicating the alert or vector status. A vectored status must be updated at the end of each update period. Thus the flight can oscillate between alerted and vectored status if there are communication delays. If this subfunction is not selected, no delays are calculated. If this subfunction is lost, no action is taken.

2.2.3.2.2.4 Fuel Utilization Subfunction. This subfunction aborts the aircraft if the remaining fuel is less than that required to complete the flight plus the required reserve. The subfunction is called periodically (when it is on), based on the flow rate and fuel status. The input data item is the reserve fuel requirement measured in pounds. If this subfunction is lost, the mission is aborted.

2.2.3.2.2.5 Refueling Subfunction. Refueling occurs when this subfunction is turned on. Hookup time is determined from a uniform probability distribution specified by the minimum and maximum hookup time. Several aircraft can be refueled simultaneously. The time to refuel is a function of the refueling rate and amount of fuel to be added. The input data items are:

1. Minimum hookup time,
2. Maximum hookup time,
3. Refueling rate (lb/min), and
4. Number of aircraft refueled simultaneously.

If this subfunction is lost, the mission is aborted.

2.2.3.2.2.6 Visual Detection Subfunction. The visual detection subfunction has computations in both the nominal and Monte Carlo portions of the AEP. The purpose of the nominal evaluation is to compute the cumulative probability of detection as a function of time for each aircraft. The cumulative probability is applicable as long as the opposing aircraft fly the nominal trajectory. If the opposing flight intentionally departs from the nominal, detection is computed based on the mean detection range (from a Rayleigh distribution).

The input data items for this function are:

1. Azimuth field of view search limits for each aircraft—half angle symmetrical about the nose (degree),
2. Maximum elevation search angle (degree),
3. Time for an individual scan for each aircraft,
4. Ratio of time spent searching to total time for each aircraft,
5. Meteorological visibility (nmi),
6. Background luminance (typically 1,000),
7. Target luminance (typically 2,200),
8. Sky-background brightness ratio (typically five),
9. Contrast compensation factor for observers (typically 1 for good observer, .5 for untrained observer),
10. Contrast compensation factor due to target shape, and
11. Mean detection range for off-nominal closing aircraft (nmi).

If this subfunction is lost, the flight is aborted.

2.2.3.2.2.7 Radar Detection Subfunction. The radar subfunction has computations in both the nominal and Monte Carlo portions of the AIRAEP. The program models a high PRF pulse Doppler radar. The purpose of the nominal evaluation is to compute the cumulative and single look probabilities of detection as a function of time. If the opposing flight intentionally departs from the nominal, detection time is recomputed within the Monte Carlo.

The input data items are:

1. Azimuth coverage for each aircraft-total angle-symmetrical about the nose (degree),
2. Elevation angle of the composite beam center (degree),
3. Width of the composite beam in elevation—for multiple bar search use the total width (degree),
4. Frame time—time to complete one sweep cycle (degree),
5. Dwell time—time that the beam would illuminate a point in space as it sweeps across the point (s),
6. Probability of the radar operator observing a single scan, and
7. Terrain reflectivity (typically .1).

This subfunction uses several other data items entered under the radar and main beam

AD-A055 591 RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 1/3
AFAL SIMULATION FACILITY/CAPABILITY MANUAL. VOLUME I. EXECUTIVE--ETC(U)
JUN 77 R A WHISNANT, W H RUEDGER, R L EARP F33615-76-C-1308
UNCLASSIFIED AFAL-TR-77-118-VOL-1 NL

3 OF 5
AD A055 591

clutter filter equipment items. In addition, it uses selected tables of aircraft radar cross section. If this subfunction is lost, the mission is aborted.

2.2.3.2.2.8 IR Detection Subfunction. The infrared detection subfunction has computations in both the nominal and Monte Carlo portions of the AIRAEP. The purpose of the nominal evaluation is to compute the cumulative probability of detection as a function of time for each aircraft. The cumulative probability is applicable as long as the opposing aircraft fly the nominal trajectory. If the opposing flight intentionally departs from the nominal, detection is computed based on the mean detection range (from a Rayleigh distribution).

The input data items for this function are:

1. Upward elevation limit (degree),
2. Downward elevation limit (degree),
3. Frame time (s),
4. Collecting aperture area of the optics (cm^2),
5. Focal length (cm),
6. Instantaneous detector FOV (degree),
7. Electrical filter factor (typically near 1.0),
8. Probability of an operator observing a detection,
9. False alarm number (typically 10.**6),
10. Atmospheric model,
11. Haze,
12. Visual range at sea level (km), and
13. Target radiant intensity (watts/cm^2/ster).

If this subfunction is lost, the mission is aborted.

2.2.3.2.2.9 Target Identification Function. This function is called after detection has been completed. If this function is not selected by the user, identification is assumed not to be required and the program immediately sequences to the engagement function. Identification can occur via an electronic IFF, TV display, or visual subfunction. The electronic IFF subfunction draws a random number for each aircraft to determine whether IFF is possible. If the draw is affirmative, another number is drawn from a Rayleigh distribution to determine the time required for identification. The TV and visual identification subfunctions require the user to specify a mean identification range. This is used to describe a Rayleigh distribution from which the identification time can be determined.

The input data items for the electronic IFF are:

1. Probability that identification can be achieved,
2. Mean delay (s), and
3. Maximum delay (s).

The input data item for the TV and visual subfunctions is the mean identification range (nmi). If any of these subfunctions are lost the mission is aborted.

2.2.3.2.2.10 Engagement Function. The engagement is halted whenever both sides have detected each other. When a weapon reaches the target flight, it is assumed that that flight detects the attacker. After the engagement is halted, all of the airborne weapons are processed to determine their effect. However, no additional weapons are fired. At the beginning of the attack, the attacking flight heads directly toward the target flight.

The input data items for both the radar and IR missile engagement are:

1. Maximum firing range (nmi),
2. Minimum firing range (nmi),
3. Maximum azimuth launch envelope (degree),
4. Maximum upper launch envelope (degree),
5. Maximum lower launch angle (degree),
6. Number of missiles,
7. Number of missiles per salvo,
8. Time between salvos (s),
9. Initial launch delay (s),
10. Missile reliability,
11. Probability of target kill, and
12. Missile velocity (kn).

If either subfunction is lost, the mission is aborted.

2.2.3.2.2.11 Formation Subfunction. The purpose of this function is to specify the position of flight members relative to the lead aircraft. The user specifies the distance right, behind, and above the leader for up to three supporting elements.

2.2.3.2.2.12 Weaving Escort Maneuver Subfunction. The purpose of the maneuver function is to define characteristics of specific maneuvers that are not incorporated in the

nominal flight profile definition. This subfunction creates a weaving escort pattern. This pattern is used to support a slower moving aircraft. The user specifies the width of the pattern (nmi) and the mean velocity of the aircraft being escorted (kn). The program then internally generates new waypoints based on the difference between the fighter aircraft velocity and mean velocity. If the flight is composed of more than two aircraft, then two aircraft fly a mirror image of that pattern. The reason for providing this function was to preclude requiring users to define the internally generated waypoints where repeating patterns were desired.

2.2.3.2.2.13 Weapon Detection Function. Weapon detection occurs automatically if a missile intercepts the target flight. The purpose of adding this function is to allow a modified kill probability if the target flight detects the approaching missile early enough. However, at present, a single kill probability is used regardless of when detection occurs. The input data items are:

1. Probability of missile detection, and
2. Mean detection range (nmi).

If this subfunction is lost, the mission is aborted.

2.2.3.2.2.14 Mandatory Operations Function. This is a special function used to define critical elements of the aircraft. If the first subfunction is lost, the aircraft aborts. If the second is lost, the aircraft is lost.

2.2.3.2.3 Modes and states

A mode is defined as a suite of subsystems which allows a particular function to be performed. Several operating modes are possible for each function or subfunction (primary and backup operating modes). The concept of modes is quite straightforward for a one-aircraft simulation. In that case, there is a suite of hardware and performance data associated with each mode. Presumably, a user sets up a problem such that the first mode represents the best performance and subsequent modes represent degraded performance with less demanding hardware requirements.

The introduction of multiple aircraft complicates the problem. Modes must now apply to the complete flight. Thus, when entering data for a mode belonging to the radar detection subfunction, the user specifies data applicable to all aircraft in the flight. Some of

the data are the same for all aircraft in the flight. But other data, such as field of view and frame time, must be specified for each aircraft, all as part of the same operating mode. A backup mode may be required if one of the aircraft loses a hardware item or aborts the mission. The user defines mode regression criteria using subfunction states. A subfunction state defines the equipment status for each aircraft for that subfunction. The user defines states for a single aircraft for each selected subfunction. Associated with each state is a suite of hardware selected from the available equipment candidates. Based on the definition of these states, the user uses Boolean and/or logic to define criteria for being in each mode. For example, mode 1 of a given subfunction may require that aircraft A, B, and C be in the primary state and that aircraft D be no lower than state 3. If this criteria was not satisfied because of failure of some equipment, mode regression would occur.

2.2.3.3 Aircraft Equipment

Aircraft hardware data are specified by the establishment of sections and candidates. A section is a particular type of hardware such as airframe, propulsion, ordinance, UHF radio, inertial navigation system, etc. Within each section the user can specify several candidates. In the section called inertial navigation systems, there could be an LN-15, LN-12, INS-61, etc., reflecting specific inertial navigation systems. The user usually has complete flexibility to aggregate or disaggregate actual black boxes into equipment elements by defining the necessary candidates for each section and creating the appropriate states and mode requirements under the Aepdeck command. To define a candidate, values are entered for the data item associated with the section.

Following is a list of the standard data items associated with every section in the air-to-ground program:

1. MTBF Mean time between failures based on flight hours,
2. MTBMA Mean time between unscheduled maintenance actions,
3. OFR Operational hours per flight hour,
4. PV Vulnerability factor,
5. NR Number of redundant boxes,
6. MTTR Mean time to repair,
7. PR Probability the box will be replaced,
8. PA Probability the replacement box is available,
9. PU Probability of undetected failure,
10. PF Probability of false failure,
11. AC Acquisition cost, and
12. UMC Cost per unscheduled maintenance action.

Two sections have additional data items which reflect performance data unique to those sections. (See equipment subsection titled special sections). In general, however, the above items relating to hardware reliability and cost are the only items associated with each section. The first two digits of the standard Air Force work unit code (WUC) identify a section.

A candidate cannot be created unless the section with which it is to be associated has been created previously. The following information is required for each candidate:

1. Name of the candidate,
2. Number of the section with which it is associated, and
3. Values for the data items.

The following six data items are associated with each section in the air-to-air program:

1. Mean time between failures,
2. Mean time to replace,
3. Repair time,
4. Mean time between maintenance actions,
5. Warmup time, and
6. Vulnerability factor.

Several sections require additional data items which are unique to those sections.

The data items which are applicable to a candidate were specified when the section was created. Once the appropriate section has been identified, the user can obtain a listing of the data items to determine the data required.

The following paragraphs briefly describe existing special sections resident in AEP.

2.2.3.3.1 Special air-to-ground sections

2.2.3.3.1.1 Airframe Section. This section has extra items used to define aircraft data required to simulate the vehicle equations of motion. Several of the aerodynamic derivatives are a function of mach number. The user can enter up to four values which the program uses to develop the continuous curve of variable versus mach number. The mach numbers for the four entries are: (1) takeoff—Mach 2, 3, (2) subsonic—mach 0.9, (3) transonic—mach 1, 0, and (4) supersonic—mach 1, 1. If the aircraft does not operate above mach 0.9, then the

transonic and supersonic values need not be entered. However, the user should be careful that the flight profile does not inadvertently cause the aircraft to enter this flight regime.

For a development of the equations of motion and more detailed definition of the data required, see AFAL-TR-73-44, Pages 10-17. In addition, volume II of that report (classified confidential) contains an unclassified example of a development of the required data for an F-4.

2.2.3.3.1.2 Propulsion Section. This section contains extra items used to enter data regarding engine characteristics. There are 4 items over and above the standard 12 items that must be specified. These are:

1. Military specific impulse (hr)—this item is used to calculate fuel consumption for normal throttle settings;
2. Maximum specific impulse (hr)—this item is used to calculate fuel consumption when the afterburner is used;
3. Military thrust (lb)—maximum thrust without the use of afterburner; idle thrust is assumed to be 20 percent of this value; and
4. Maximum thrust (lb)—maximum thrust with afterburner; if the aircraft does not have an afterburner, items 2 and 4 above should be the same as 1 and 3, respectively.

2.2.3.3.2 Special air-to-air sections.

2.2.3.3.2.1 Airframe Section. The first section is utilized to define the aircraft data required to simulate the vehicle equations of motion. Several of the aerodynamic derivatives are a function of mach number. The user can enter up to four values which the program uses to develop the continuous curve of variable of interest versus mach number. The mach numbers for the four entries are: (1) takeoff—mach 0.3, (2) subsonic—mach 0.9, (3) transonic—mach 1.0, and (4) supersonic—mach 1.1. If the aircraft does not operate above mach 0.9, then the transonic and supersonic values need not be entered. However, the user should be careful that the flight profile does not inadvertently cause the aircraft to enter this flight regime.

2.2.3.3.2.2 Propulsion Section. The second section is utilized to enter data regarding engine characteristics. Four items over and above the standard six items must be specified. These are:

1. Military specific impulse (hr)—this item is used to calculate fuel consumption for normal throttle settings;

2. Maximum specific impulse (hr)—this item is used to calculate fuel consumption when the afterburner is used;
3. Military thrust (lb)—maximum thrust without the use of afterburner; idle thrust is assumed to be 20 percent of this value; and
4. Maximum thrust (lb)—maximum thrust with afterburner.

If the aircraft does not have an afterburner, Items two and four above should be the same as one and three, respectively.

2.2.3.3.2.3 Radar Section. This section is reserved for pulse Doppler radar. The peculiar data items are:

1. Peak transmitted power (W),
2. Pulse repetition frequency (Hz),
3. High (1) or low (2) PRF (not presently used),
4. Transmit duty cycle,
5. Receive duty cycle,
6. Antenna gain (db),
7. Wavelength (m),
8. Bandwidth (Hz),
9. Noise figure (dB),
10. System losses (dB),
11. Sidelobe gain (dB), and
12. False alarm number.

2.2.3.3.2.4 Radar Main Beam Clutter Filter Section. This section is reserved for the radar main beam clutter filter data. Data are entered as filter gain (dB) versus the difference between range rate and clutter velocity (FPS). Up to four values of gain can be entered (all negative dB) for four values of velocity difference (must be positive monotone increasing).

2.2.3.3.2.5 IR Detector Section. This section is reserved for entering the response versus wavelength (micron). Up to eight values of response can be entered. The units of detectivity are CM*HZ**.5/W. The logarithm (base 10) of detectivity is entered. The data couples should be entered in increasing values of wavelength.

2.2.3.3.2.6 IR Optics Section. This section is reserved for entering the IR optics transmissitivity versus wavelength (μ). Up to four data couples can be entered in increasing values of wavelength. The transmissitivity can take on values between zero and one.

SECTION 2.2 BIBLIOGRAPHY

AEP Programs

Battelle. Columbus, *Avionics Evaluation Program - User Manual*, July 19, 1976.

Battelle. Columbus, *Air-to-Air Avionics Evaluation Program - User's Manual*, July 31, 1975.

Battelle. Columbus, *Avionics Evaluation Program: Multiple Aircraft, Multiple Sorties, and Cost Accumulation*. AFAL-TR-76-196, November, 1976.

Summers, Diane E., and Welp, David W. "Aviation Evaluation Programs." published paper, source unknown.

Welp. *Application of Interactive Graphics to the Avionics Evaluation Program*. AFAL-TR-73-270, November 1973.

Welp. *A Computer Program for Simulation and Effectiveness Evaluation of Avionics for Military Aircraft*. AFAL-TR-73-44, February 1973.

2.3 GASP IV

2.3.1 Models, Systems, and Simulations

A simulation language provides the structure and terminology to facilitate the building of simulations. GASP IV is such a computer language; it helps the user to build computer simulation programs that can be both the model of the system and the analysis vehicle. Thus this program can be considered a model of a system and a generator of statistical data about the model of the system.

GASP IV enables statistical experiments to be conducted, but it does not deal with actual experimental design. GASP IV can make simulation economical and technically feasible, but it does not provide information that makes the "simulate or not" decision any easier.

2.3.1.1 Features

As a programming language, GASP IV gives the computer programmer a set of FORTRAN statements designed to carry out the most important functions in simulation programming. Modeling concepts are translated by GASP IV into FORTRAN routines that can be easily used. GASP IV has five distinct features that make it particularly attractive as a simulation language:

1. GASP IV is FORTRAN based and requires no separate compiling system.
2. GASP IV is modular and can be made to fit on all machines that use a FORTRAN IV compiler.
3. GASP IV is easy to learn since the host programming language, FORTRAN IV, is usually known, and only the simulation concepts need be mastered.
4. GASP IV can be used for discrete, continuous, and combined modeling.
5. GASP IV is easily modified and extended to meet the needs of particular applications.

2.3.1.2 Discrete, Continuous, and Combined Simulation

Simulation is divided into two categories: discrete change and continuous change. These terms describe the model, not the real system. In fact, it may be possible to model the same system with either a discrete change (hereafter referred to simply as discrete) or a continuous change (continuous) model. GASP IV is designed to accommodate both categories of models, separately or combined. In most simulations, time is the major independent variable. Other variables included in a simulation are functions of time and are the dependent variables. The adjectives discrete and continuous refer to the behavior of the dependent variables.

Discrete simulation occurs when the dependent variables of the model change discretely at specified points in simulated time. The time variable may be either continuous or discrete in such a model, depending on whether the discrete changes in the dependent variables can occur at any point in time or only at specified points. Variable time increment (including next-event) procedures result in a continuous time variable, whereas fixed time increment procedures are normally discrete in time. GASP IV is a discrete simulation language; however, it is necessary to recognize that a digital computer is technically discrete in its operation. As a practical matter, however, any variable whose possible values are limited only by the inherent capabilities of a computer is considered continuous.

In continuous simulation, the dependent variables of the model may change continuously over simulated time. A continuous model may be either continuous or discrete in time, depending on whether the values of the dependent variables are available at any point in simulated time or only at specified points in simulated time.

In combined simulation the dependent variables of a model may change discretely, continuously, or continuously with discrete jumps superimposed. The time variable may be discrete or continuous.

GASP IV is a language that can be used for discrete, continuous, or combined simulation. In GASP IV, the most important characteristics of combined simulation, which arise from the interaction between discretely and continuously changing variables, are easily modeled. In general, this interaction takes one of three forms. First, discrete changes may be applied to "continuous" variables. Second, achieving specified conditions for a state variable may cause an event to occur or to be scheduled. Third, the functional description of continuous variables may be changed discretely.

2.3.2 GASP IV Philosophy

The philosophical basis for the design of GASP IV is the concept of modeling a system in two dimensions: the time dimension and the state-space dimension. Fundamental to building a GASP IV simulation model is the decomposition of time and the state space into manageable elements. The decomposition in the time dimension involves the defining of events and potential changes to the system when an event occurs. The GASP IV philosophy requires that the user specify the causal mechanisms by which events can occur, but relieves him completely of the need for sequencing the events. Thus the user must only define the mathematical-logical relations that transpire at an event occurrence, and he is not required to model the timing of events during a simulation. This decomposition of the time axis into points at which events could occur is a huge reduction in the size of the modeling effort.

In the state-space dimension GASP IV presumes that a system model can be decomposed into its entities, which are described by attributes. These are further classified as discrete or continuous. The use of the adjectives "discrete" and "continuous" is motivated by their use in the definitions of discrete and continuous simulation. More descriptive adjectives would be static and dynamic. The value of a discrete attribute remains constant between event times. The value of a continuous attribute may change between events according to a prescribed dynamic behavior. Because of the special nature of continuous attributes and the need for the user to model their dynamic behavior, they are referred to as state variables. Special storage arrays are provided in GASP IV for storing values of state variables and, also, if required, their derivatives and immediate past values. To avoid confusion, the word "attribute" is used to refer only to a discrete attribute.

GASP IV specifies that the status of a system be described in terms of a set of entities, their associated attributes, and state variables. The GASP IV simulation philosophy is that a dynamic simulation can be obtained by modeling the events of the system and by advancing time from one event to the next. This philosophy presumes a broader definition of event than has normally been used in discrete-event languages:

An event occurs at any point in time beyond which the status of a system cannot be projected with certainty.

Events usually cause changes in the status of the system or in the equations defining the state variables of the system. However, this definition does not necessarily relate an event to any change, either discrete or continuous, in the status of a system. Events could occur at decision points where the decision is not to change the status of the system. Conversely, the above definition allows the system status to change continuously without an event occurring, as long as the change has been prescribed in a well-defined manner.

In GASP IV it is useful to describe events in terms of the mechanism by which they are scheduled. Those that occur at a specified projected point in time are referred to as time-events. They are commonly used in conjunction with "next event" simulation. Those that occur when the system reaches a particular state are called state-events. Unlike time-events, they are not scheduled in the future but occur when state variables meet prescribed conditions. In GASP IV, state-events can initiate time-events and time-events can initiate state-events.

The behavior of a system model is simulated by computing the values of the attributes at event times. The time step increment is automatically determined by GASP IV, based on the equation form for the state variables, the time of the next event, and accuracy and output requirements.

When an event occurs, it can change system status in three ways: by altering the value of state variables or the attributes of the entities; by altering relationships that exist among entities or state variables; and/or by changing the number of entities present. Methods are available in GASP IV for accomplishing each type of change. Any of these changes can result from the occurrence of an event. Between event times, only the values of the state variables can change.

At each time-step, the state variables are evaluated to determine if the conditions prescribing a state-event have occurred. If a state-event was passed, the step size was too large and is reduced. If a state-event occurs, the model status is updated according to the user's state-event subroutine. Step size is automatically set so that no time-event will occur within a step. This is accomplished by setting the step size so that the time-event ends the step.

Since time-events are scheduled happenings, certain attributes are associated with them.

At the minimum, a time-event must have attributes that define its time of occurrence and its type. If the time-event is associated with an entity, either the attributes of that entity must be associated with it or the event must be able to refer to the attributes of the entity. For example, if there is an end-of-service event for an item, the attributes for that item must in some way be associated with the event.

2.3.2.1 Data Storage and Timing Requirements

In GASP IV a system's file-entity-attribute structure is established by data declarations. Entities are represented by vectors and matrices, the elements of which are stored and represent attribute data.

The membership of entities in groups (such as a waiting line) is changeable, and the data that express them are stored differently. They are stored in computer lists called files. A single array NSET is used to store all files. Special routines, such as FILEM (to put an entity in a file) and RMOVE (to take an entity from a file), are part of the GASP IV language.

Entities, attributes, state variables, and file memberships comprise the static structure of a simulation model. They describe the state of a system model but not the operation of the system. The latter depends on event definitions and the equations defining the behavior of the state variables.

The key to event simulation is the ability to organize events so that they are executed within the computer in an order corresponding to that which would occur in the real system. This preserves the time relationship between simulated and real events. Ordinary programming languages are unsuitable for this task because they operate in a strictly sequential manner; there is no way to tell a FORTRAN program to "do something later" without building special subprograms. GASP IV provides these subprograms.

2.3.2.2 Method of Simulation Programming

Every GASP IV simulation model consists of: (1) a set of event programs or state variable equations, or both, that describe a system's dynamic behavior, (2) lists and matrices that store information, (3) an executive routine that directs the flow of information and control within the model, and (4) support routines. These form an operating computer program whose performance reflects that of a simulated system. A GASP IV program is made up of subprograms linked together by an executive routine that organizes and controls the performance of the subprograms.

Simulation programs contain routines for reading input data, performing record keeping tasks, advancing time, updating system status, producing reports of the simulated system's behavior, and so forth. A simulation program is said to be in a particular mode of operation when it is performing the corresponding function, for example, the input mode or the simulation mode. A mode can, when necessary, call upon many separate programs, each performing a task related to the general function of the mode.

2.3.2.3 GASP IV Functional Capabilities

GASP IV is organized to provide eight specific functional capabilities:

1. Event control.
2. State variable updating using integration if necessary.
3. Information storage and retrieval.
4. System state initialization.
5. System performance data collection.
6. Program monitoring and event reporting.
7. Statistical computations and report generation.
8. Random deviate generation.

Four of these capabilities, (1), (2), (4), and (6), are primary functions. At the time a program is engaged in one of them it is not performing any of the others. The remaining four, (3), (5), (7), and (8), are support oriented; they provide a fundamental computational or data processing capability that is not control oriented.

The four primary capabilities constitute the basic modes of GASP IV, which are shown in Figure 2.3.2.3-1.

Two forms of program control are required. One directs the program into its various modes: initialization, state variable updating, monitoring, and so forth. The other operates within the simulation model and sequences the execution of event routines. In GASP IV, the first function is called the executive function and the second is called the event selection function.

The executive function is computationally and logically oriented; it switches from mode to mode as the logic of a program dictates. It also updates state variables over time using a step-evaluate-step procedure. At each step, the state variables are updated and their values are checked against the conditions defining state-events. The event-selection function is time-oriented; it switches to an event when one occurs.

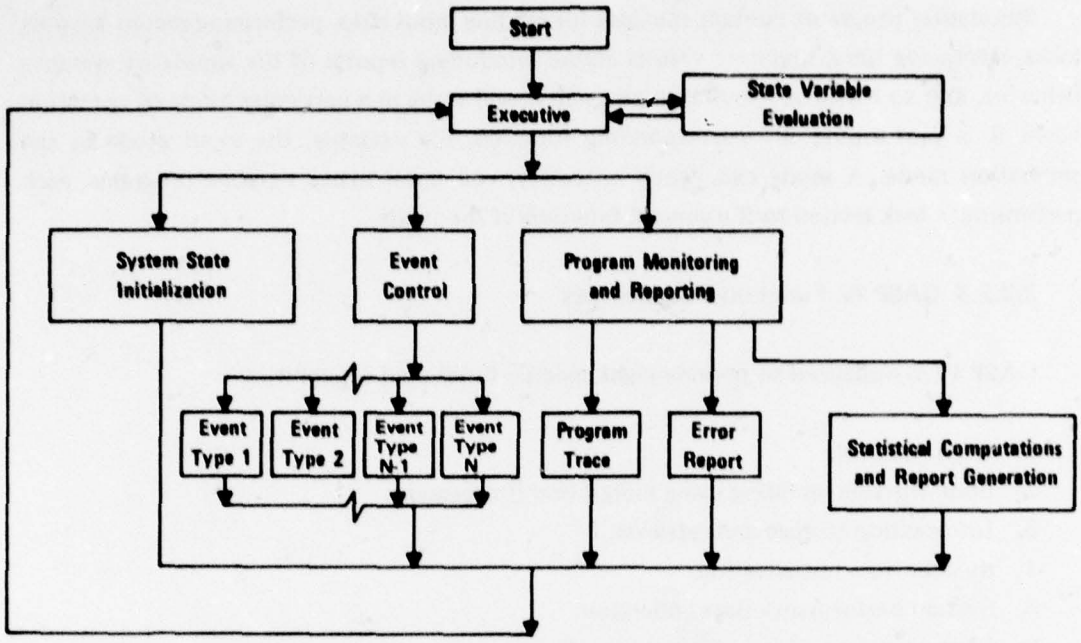


Figure 2.3.2.3-1. Basic modes of GASP IV control.

Additional simulation flexibility can be achieved by defining control events in addition to system activity events. These events are time oriented instructions to the executive that allow mode switching to be performed on a time and a logic basis. One can schedule an event to "go into output mode" when the simulator clock reaches 1000 hours as well as when demand exceeds 100 units.

The form of organization used in GASP IV is hierarchical; there are two levels of control, and each level has authority over its subordinate levels.

The highest level of program control, the executive level, determines what the program must do at each point in simulated time and directs control to the appropriate mode to perform the selected task. Control passes from executive level to mode and back again. Control is maintained within a mode until all computations and evaluations pertinent to that mode have been made. The sequencing of the computations and evaluations is accomplished as part of the function designed into the subprograms of the mode.

One of the most important uses of simulation is to investigate the effect of changes in a system on selected measures of system performance. This is true whether a simulation is

being used for evaluation or design. The "hierarchical control-mode-data pool" concepts provide the needed flexibility in design and adaptation, since changes in simulation models usually require changes in input data, system elements, or event logic. A modular structure allows event logic to be changed quite simply, because each event is a separate subprogram. A data pool allows changes made in data inputs to be communicated throughout an entire simulation model by a change in a single data input. The preparation of reports summarizing the results of simulation runs is simplified by utilizing standard report programs that obtain their information from the common data pool. Perhaps most important of all, model debugging is simplified by providing access points at which program results can be sampled without interfering with the logic of particular events.

2.3.3 GASP IV Definitions and Procedures

A simulation program written in GASP IV consists of two parts: a user part and a GASP IV part. The user part contains subprograms for initialization (the main program and subroutine INTLC); equations for state variables and conditions defining state-events (subroutines STATE and SCOND); event code definitions (subroutine EVNTS); event processing (subroutines called by EVNTS); and special data collection and reporting procedures (subroutines SSAVE, OTPUT, and UERR). The GASP IV part contains subprograms that provide for the following functions: the executive or mode controller (subroutine GASP), data and event initialization (subroutine DATIN), data storage and retrieval, data collection, statistics computation and reporting, monitoring and error reporting, random deviate generation, and miscellaneous support.

The GASP IV function of the status advance includes the sequencing of time-events, identification of state-events, and state variable integration if needed. It is the heart of the executive process. The main program initializes the non-GASP variables that are to remain constant for all simulation runs. Subroutine GASP is then called from the main program. The general layout of the main program is shown in Figure 2.3.3-1. Subroutine GASP's first action is to call subroutine DATIN, which initializes all GASP IV variables either through arithmetic statements or the reading of data cards. Either data cards or a programming code can be used to establish the initial events and entities for the simulation. Subroutine DATIN calls the user generated subroutine INTLC, which is used to initialize non-GASP variables at the start of each run. Through the use of subroutine INTLC the user can perform sequential simulations with changes in the non-GASP variables. Initialization of state variables is accomplished by a call to subroutine STATE. This completes the initialization phase of the simulation.

```
C     MAIN PROGRAM
      DIMENSION NSET (NNSET)
C     NNSET to be specified
      COMMON (GASP variables)
      COMMON (non-GASP variables.  This must be a named
      COMMON Block)
      EQUIVALENCE  (NSET(1), QSET(1))
C     Initialization of non-GASP variables
C     Initialize Card Reader Value, NCRDR and Printer Value,
C     NPRNT.
      CALL GASP
C     If more runs are desired, insert GO TO statement
C     to either reinitialize non-GASP variables or
C     to CALL GASP again
      STOP
      END
```

Figure 2.3.3-1. Layout of main program.

During a simulation run, subroutine GASP determines if a time advance by a step or to the next event should be made. If the simulation only involves time-events, TNOW is set equal to the time of the next event. If only a continuous simulation is being performed, TNOW is advanced to the time at which the next evaluation of the state variables is to be made. If a combined simulation is being performed, the time advance occurs by steps, with the step size adjusted when necessary to end at an event. A run can be completed either by setting a time for completion or by having an end-of-simulation event. If a run is not completed, the clock for the simulation, as represented by the variable TNOW, is advanced.

In continuous and combined simulations, the updating of the state variables is performed in subroutine STATE. The vectors SS(·) and DD (·) are used to define the state variable values and their derivatives, respectively. The user must code subroutine STATE in a manner that permits the calculation of SS(·) or DD(·). Once the state variables are updated, their values are checked against user prescribed conditions that define state-events in subroutine SCOND. If no state-event has occurred, time is advanced again by subroutine GASP.

If time has advanced to a time-event, subroutine GASP obtains the attributes for the event by removing the first event stored in file 1, which is the event file or calendar of

events. The user written subroutine EVNTS(IX) is then called to decipher the event code, IX, and to call the appropriate event subroutine.

A call to subroutine EVNTS is also made if a state-event occurs. In this case, IX is the state-event code indicating that a state-event has occurred.

When a simulation run is completed, subroutine OTPUT is called. Subroutine OTPUT provides the user with a mechanism to make final calculations for the simulation run and to print out any specific information that was collected during the run. Thus the call of subroutine OTPUT can be used as an end-of-simulation event. Following a return from subroutine OTPUT, subroutine GASP calls subroutine SUMRY, which prints out the final GASP IV summary report, including statistical information, tables of state variables, and plots of state variables. The user can also suppress the printing of the GASP IV summary report. Following the printing of the final summary report, tests are made to determine if other simulation runs are to be made. A new simulation run can be started by returning to the main program for complete reinitialization or to subroutine DATIN for partial reinitialization. If another run is to be made, the above-mentioned process is repeated. If no further runs are required, the simulation program is terminated.

2.3.3.1 An Overview of Subroutine GASP

Figure 2.3.3.1-1 presents a descriptive flow chart of subroutine GASP. GASP first calls subroutine DATIN, which initializes all GASP IV variables either directly or from reading data cards. Initial events and other file entries can also be established by DATIN. Besides initialization, DATIN also prints out the values of the input data. Events to occur at the beginning of the simulation are then processed.

A test is then made to determine if the simulation involves only time-events, that is, a discrete simulation. If this is the case, a next event time advance is used and the simulation proceeds from time-event to time-event until a signal is given to end the simulation. The method used in GASP IV to perform the next event portion of a simulation involves file 1, the event file used to store attributes associated with events. Since events must be inserted in the file and removed from it in chronological order, it is necessary to have routines for these functions. The GASP IV subroutine FILEM puts events in the event file based on the time-of-event attribute. The subroutine RMOVE removes events from the event file. Thus GASP IV provides an information storage and retrieval system for events.

In addition to the calendar of events, other files of information must usually be

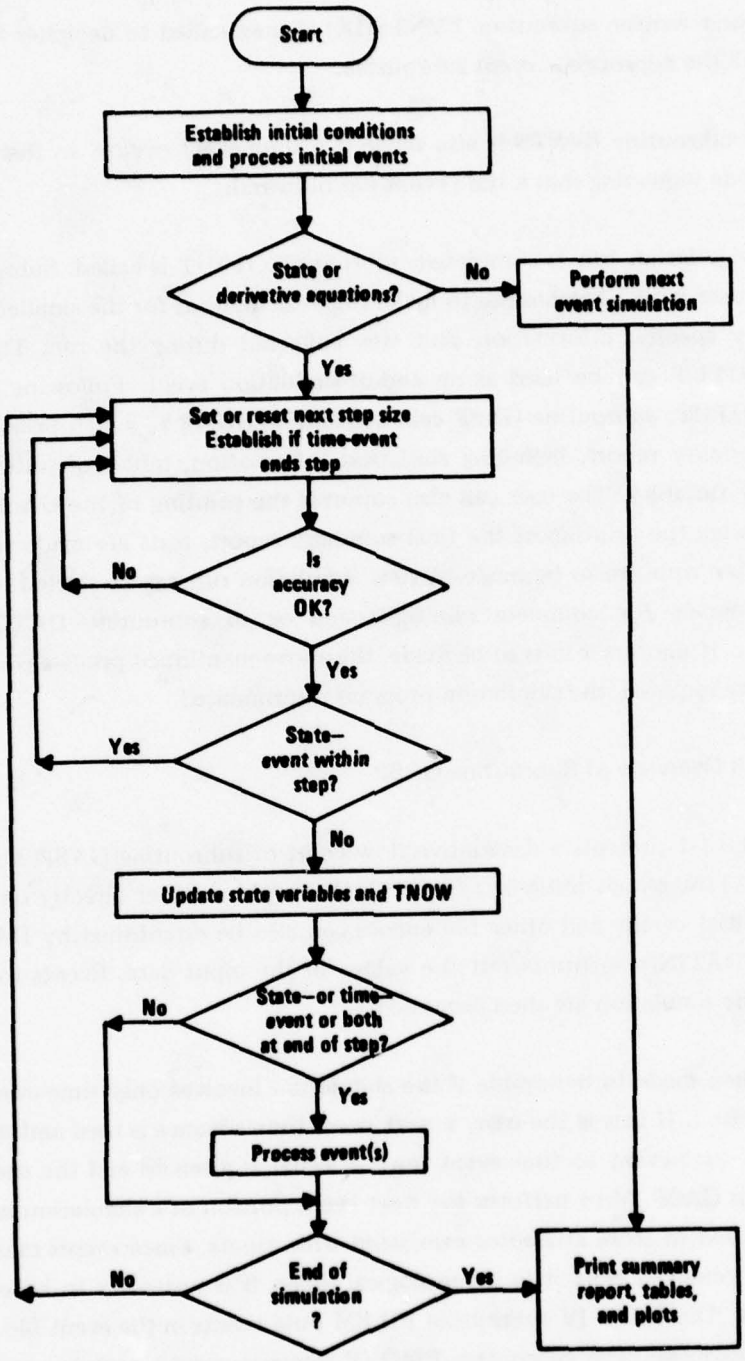


Figure 2.3.3.1-1. Descriptive flow chart of subroutine GASP.

maintained during simulation runs. All files are stored in the equivalenced arrays NSET/QSET and are processed by the subroutines FILEM and RMOVE.

To write a discrete-event simulation program, it is necessary to specify the changes that occur at event times, and the future events that are generated by event occurrences. The user writes these event routines, such as ARRIV and ESERV, which specify what happens in the simulation model when these events occur.

To write a continuous simulation program, the equations governing the dynamics of the state variables must be defined, and state-event conditions must be specified if they exist. For a combined discrete-continuous simulation, all of these must be written. GASP IV has been designed to reduce, as much as possible, the amount of programming necessary beyond writing the event and state variable routines.

If the simulation involves "continuous" variables, a step-evaluate-step time advance procedure is used. The step size is variable to ensure that no events occur within the step and that the desired accuracy in the calculation of state variables is maintained. The accuracy test is necessary only if state variables are described in terms of a derivative equation, where integration is required to obtain state variable values. To ensure that a time-event does not occur within a step, the step size is automatically adjusted so that the time-event occurs at the end of the step. For state-events the step size is reduced if such an event occurs within the step. The step size is set so that either no state-event occurs within it (but may occur at the end of the step) or the minimum step allowed is used. The minimum step allowed is specified as an input value by the user. A state-event occurring within the minimum step size is considered to occur at the end of the step.

The conditions for a state-event are defined by the user and would normally involve a state variable achieving a threshold with a prescribed tolerance. The user codes these conditions in subroutine SCOND. The equations defining the state variables are coded in subroutine STATE. Subroutine GASP establishes a step size by which it attempts to advance time. Subroutine STATE is called to compute the values of the state variables at the end of the proposed step. For each derivative equation, intermediate calls to STATE are made to perform accuracy checks and to allow derivative equations to be functions of the state variables. If accuracy is not met, the step size is reduced accordingly and the process described above is reinitiated.

If accuracy conditions are met, subroutine SCOND is called to check whether a

state-event occurred within the step. After a return from SCOND, GASP performs a test on the GASP IV variable, ISEES, to determine if a state-event has been passed (ISEES $< 0.$), if one occurs at the end of the step (ISEES > 0), or neither of these (ISEES = 0). In the first case, the step size is reduced and the above-mentioned process is repeated. In the latter two cases, the step size is accepted and model status, including TNOW, is updated. GASP IV provides the subprogram KROSS, which automatically sets ISEES. Function KROSS detects the crossing of a variable beyond a threshold, or the crossing of one variable by another variable. If the user does not employ KROSS, he must set ISEES before returning from subroutine SCOND.

If the step ends with an event, this is processed by calling subroutine EVNTS, which calls the appropriate event based on the event code of the event occurring. A check is then made to determine if the simulation is to be ended. If not, a next step is processed.

At the end of the simulation run, a standard GASP IV summary report is printed, and tables and plots of the state variable values are printed as requested.

Table 2.3.3.1-1 lists definitions for the GASP IV variables that are commonly used by the GASP IV programmer. Other important variables are established through data input and these are defined in Table 2.3.5.1-1.

All GASP IV variables, with the exception of NSET, are in named COMMON blocks. Blank COMMON is only used for the array NSET. The GASP IV user must not use blank COMMON for non-GASP variables.

2.3.3.2 Model Status Definition and Control

The status of a GASP IV simulation model is defined in terms of events, entities and their attributes, state variables, or a combination of the three. Entity definition and storage in GASP IV is accomplished through the use of the file storage arrays NSET/QSET and file processing routines. Events are scheduled to occur either at a future time (a time-event) or when state variables meet specified conditions (a state-event). Time-events and their associated attributes are stored in file 1 using predecessor and successor pointers. Entities and their associated attributes are stored in file 2 through file NNFIL. State variables and their derivatives are stored in the GASP IV arrays SS(·) and DD(·).

2.3.3.3 State Variable Definition

The GASP IV variables SS(I), SSL(I), DD(I), and DDL(I) are used to define the state

TABLE 2.3.3.1-1. DEFINITIONS OF COMMONLY USED GASP IV VARIABLES

Variable	Definition
ATRIB(I)	Buffer storage for the Ith attribute value to be stored in or removed from QSET
DD(I)	The value of the derivative of SS(I) at TNOW
DDL(I)	The value of the derivative of SS(I) at TTLAS
DTFUL	Full step size
DTNOW	TNOW-TTLAS
IIEVT	Code of state-events
IINN(I)	Priority ranking indicator for file I
ISEES	Internal indicator that a state-event ends current-step
JEVNT	Code of time-event to be processed, which is the second attribute of an event
KKRNK(I)	The attribute on which file I is ranked
LFLAG(I)	State condition flags
MFA	Relative address of the first cell of NSET/QSET available for storing a new entry
MFE(I)	The pointer to the first entry in file I
MLE(I)	The pointer to the last entry in file I
MSTOP	Indicator for specifying method of ending the simulation
NNATR	Number of attributes per entry
NCRDR	Number of the card reader (input tape number)
NFLAG	Number of state condition flags
NPRNT	Number of the printer (output tape number)
NNQ(I)	The current number of entries in file I
NSET(I)	Integer representation of filing array; used for pointers only
PPARM(I, J)	Array for storing parameter values
QSET(I)	Real valued representation of the file storage area; used for all attribute values
QQTIM(I)	Time of last use of file I
SS(I)	The value of the state variable SS(I) at TNOW
SSL(I)	The value of the state variable SS(I) at TTLAS
TTLAS	The last time at which an event could have occurred; when a step advance is in progress. TTLAS is the time at which acceptable values of SS(I) were computed
TNOW	Current time of simulation; when a step advance is in progress, TNOW is the time to which GASP is trying to advance

variables and their derivatives at times TNOW and TTLAS where

TNOW = time at which values of the state variables are being computed.

TTLAS = time at the beginning of the current step (the time at which the values for the state variables were last accepted).

SS(I) = value of state variable I at time TNOW.
SSL(I) = value of state variable I at time TTLAS.
DD(I) = value of the derivative of state variable I at time TNOW.
DDL(I) = value of the derivative of state variable I at time TTLAS.

The equations for calculating the values of SS(I) and DD(I) must be written by the user and included in subroutine STATE. All state variables are in GASP IV common storage and are accessible to the user in any subprogram.

2.3.3.4 Time Advance Procedures

In GASP IV, the amount by which simulated time is advanced depends on the type of simulation being performed and the values of specific variables at the current point in time.

2.3.3.4.1 Discrete simulation

In a discrete simulation, time is advanced from one event to the next event. It is assumed that the system status has remained constant between events.

2.3.3.4.2 Continuous simulation

For a continuous simulation, the maximum step size prescribed by the user is employed, unless an event occurs within the step. Should this occur, the step size is reduced, if the event causes a state variable to cross a threshold beyond allowable tolerances. The step size is reduced (typically halved) based upon allowable error specifications introduced by the user. When all state variables are within the accuracy specifications to a significant extent, the next step size to be used is increased by an integration algorithm.

It is obvious that the time advance procedures included within GASP IV involve many variables with many interactions between these variables. Subroutine GASP automatically advances time for the user based on the input values prescribed for the minimum step size DTMIN, the maximum step size, DTMAX, and the increment in time between the recording of the status of the system, DTSAV. The accuracy requirements of absolute error value, AAERR, and relative error value, RRERR, as well as the value of the derivative of the state variables as specified by the user also determine the time advance procedures used.

2.3.4 GASP IV Subprograms

2.3.4.1 GASP IV Storage Requirements and Limitations

GASP IV uses both named and blank COMMON storage. Variables are stored in COMMON for one of two reasons: (1) to make their values accessible to other subprograms, or (2) to prevent their undefinition upon execution of a RETURN in the subprogram in which they are defined.

Blank COMMON is only used for the array, NSET. The GASP IV user must not use blank COMMON for non-GASP variables.

NSET is placed in blank COMMON storage so that it can be dimensioned to the required size in the main program and nominally dimensioned to one in all the GASP IV subprograms. This takes advantage of the fact that the size of blank COMMON in the various subprograms comprising an executable program need not be the same. Thus precompiled GASP IV subprograms with NSET dimensioned at one can be maintained on the computing system, and the dimension of NSET can be set in the user's main program. Named COMMON is used for all GASP IV variables other than NSET, which require COMMON storage.

The organization of the named COMMON blocks was designed to minimize the number of COMMON blocks required in the user subprograms. A gross description of the variables for each common block is given below:

1. GCOM1 Variables associated with discrete simulations.
2. GCOM2 Variables associated with continuous simulations.
3. GCOM3 Variables associated with a step time advance procedure.
4. GCOM4 Variables associated with data collection and reporting.
5. GCOM5 Variables associated with run conditions and description.
6. GCOM6 Variables associated with filing system and statistical storage arrays.

The limitations imposed by array size and the variables causing the limitation are presented below. A DATA statement is included in subroutine DATIN, which specifies the largest value for each limitation. A check is made in subroutine DATIN to ensure that these array sizes are not exceeded:

1. Number of state variables ≤ 100,

2. Number of histograms ≤ 25,
3. Total number of cells for all histograms ≤ 500,
4. Number of variables for which statistics are collected ≤ 25,
5. Number of variables for which statistics are collected over time ≤ 25,
6. Number of attributes describing an event ≤ 25,
7. Number of files ≤ 100,
8. Number of parameter sets ≤ 50,
9. Number of plots ≤ 10,
10. Number of variables per plot ≤ 10,
11. Number of random number streams ≤ 6,
12. Number of state conditions ≤ 50,
13. Number of entries per file limited only by available core storage.

2.3.4.2 Functional Breakdown of GASP IV

The functional breakdown of the GASP IV subprograms is shown in Table 2.3.4.2-1. In addition to listing the general functions included in simulation programs, Table 2.3.4.2-1 categorizes each subprogram according to whether it is GASP IV provided or user written. Brief descriptions of the subroutines follow.

2.3.4.2.1 Time advance and status update (subroutine GASP)

The GASP IV function of status advance includes the sequencing of time-events, identification of state-events, and state variable integration if needed. It is the heart of the executive process.

Subroutine GASP is the GASP IV executive routine. It selects the appropriate mode of control and calls the necessary subroutines to process a simulation from initialization of the first run through output of the final run. Subroutine GASP is called only by the user written main program, and control is not returned to the main program until the requested number of runs has been completed.

2.3.4.2.2 Initialization

Initialization of variables prior to the execution of a GASP IV simulation run occurs in phases. Some variables are initialized during program compilation by DATA statements. Others are assigned values obtained directly from the reading of GASP IV input data cards. Finally, some variables are assigned initial values after all GASP IV input data cards have

TABLE 2.3.4.2-1. FUNCTIONAL BREAKDOWN OF GASP IV AND USER SUBPROGRAMS

Function	Subprograms Supporting Function	
	Provided by GASP IV	Provided by User
Time advance and status update	Subroutine GASP	Subroutine STATE Subroutine SCOND Subroutine EVNTS Event subroutines
Initialization	Subroutine DATIN Subroutine CLEAR Subroutine SET	Main program Subroutine INTLC Input data
Data storage and retrieval	Subroutine FILEM Subroutine RMOVE Subroutine CANCL Subroutine COPY	
Location of conditions and entries	Function KROSS Function NFIND	
Data collection, computation, and reporting	*Subroutine COLCT* Subroutine TIMST Subroutine TIMSA Subroutine HISTO Subroutine GPLOT Subroutine PRNTQ Subroutine PRNTS Subroutine SUMRY	Subroutine SSAVE Subroutine OTPUT
Program monitoring and error reporting	Subroutine MONTR Subroutine ERROR	Subroutine UERR
Miscellaneous support	Function SUMQ Function PRODQ Function GTABL Subroutine GDLAY	
Random deviate generation	Function DRAND Function UNFRM Function TRIAG Function RNORM Function RLOGN Function ERLNG Function GAMA Function BETA Function NPSSN Function GAM	

been read. In general, each phase is associated with specific elements of the executable program. DATA statements in individual subprograms assign values to variables local to those subprograms. The main program must initialize two GASP IV variables: the card reader number, NCRDR; and the printer number, NPRNT. The user may employ the main program to initialize non-GASP variables prior to the assumption of executive control by subroutine GASP. The preceding multiple runs are made. The last phases of initialization are controlled by subroutine DATIN; they are discussed in conjunction with the description of that subprogram.

2.3.4.2.2.1 Subroutine DATIN. This subroutine is called by subroutine GASP, and it provides for the selective initialization of GASP IV variables through four mechanisms. Some GASP IV variables are initialized to standard or default values; some are assigned values equal to or computed from GASP IV input data; some are assigned values derived from the user provided subroutines INTLC and STATE; and some are assigned values (or left unchanged) from a previous run. Subroutine DATIN accomplishes this initialization in the proper sequence to facilitate either single or multiple runs.

Subroutine DATIN also performs error checking and provides a printout (echo check) of the input data that may be selectively suppressed.

Subroutine DATIN is called only by subroutine GASP and only during the initialization phase of a simulation run.

2.3.4.2.2.2 Subroutine CLEAR. Subroutine CLEAR is used to zero out the storage arrays that collect values of variables to be reported in the final summary report. For example, if the user wants to calculate statistics based on data collected after time 1000, the statement CALL CLEAR inserted into the program at time 1000 (possibly through a time event) can be used to accomplish this. If file statistics as well as summary statistics are to be reinitialized, this can be accomplished using subroutine MONTR.

2.3.4.2.2.3 Subroutine SET. Subroutine SET initializes the filing array NSET and all variables associated with the filing array. This includes the following variables: number of entries in file I, NNQ(I); the time integrated number of entries in file I, EENQ(I); the second moment of the time integrated number of entries in file I, VVNQ(I); the largest number of entries that have been in file I, MMAXQ(I); and the last time an entry was either filed or removed from file I, QQTIM(I). A call to subroutine SET can be activated through the use of the GASP IV input cards. When subroutine SET is called, all events and entities stored in all files are deleted and the file structure is reinitialized.

2.3.4.2.3 Data storage and retrieval

The data storage and retrieval subprograms provide the mechanisms through which entries with their associated attributes are maintained. In GASP IV, entries are stored in a doubly linked list maintained in the filing array NSET. NSET is a one dimensional array that is equivalenced to the one dimensional array QSET, which is in COMMON storage. This enables each of the attribute values to be stored as a real number and pointers to be stored as integers.

Priority in a file is based on a ranking attribute and a priority method. Any attribute number can be used as the ranking attribute for a file. The ranking attribute number for file I is established by data input and is stored as the GASP IV variable KKRNK(I). Four priority codes are available in GASP IV.

1. Low-value-first (LVF).
2. High-value-first (HVF).
3. First-in-first-out (FIFO).
4. Last-in-first-out (LIFO).

2.3.4.2.3.1 Subroutine FILEM (IFILE). This subroutine stores the attributes of an entry in a file, updates statistics for the file, and maintains the time for the next discrete event TTNEX if the new entry is an event. Subroutine FILEM is a long subroutine since it contains the coding for inserting any entry into any file under any of the four priority methods.

2.3.4.2.3.2 Subroutine RMOVE. To remove an entry from a file, subroutine RMOVE is used. It is called to remove the entry whose first cell number is NTRY(I) from file IFILE. RMOVE may also be used to cancel an entry. Subroutine RMOVE is called by subroutine GASP to process the next time-event. RMOVE may also be called directly by the user.

2.3.4.2.3.3 Subroutine CANCL (NTRY). This subroutine is used to cancel an entry in the event file whose first cell is NTRY. Cancellation of an event consists of removing it from the file storage array, updating TTNEX if the event was the next event, loading the buffer ATRIB, and updating file statistics. In other words, cancellation consists of the same functions as removal.

Subroutine CANCL is included only for its mnemonic value and is equivalent to the direct use of the statement CALL RMOVE (NTRY, + 1).

2.3.4.2.3.4 Subroutine COPY (NTRY). This subroutine is called to copy the values of the attributes of entry NTRY to the buffer ATRIB. Copying an entry does not change the file storage area or its associated statistics in any way.

2.3.4.2.4 Location of state conditions and entities

Two functions, KROSS and NFIND, are included in GASP IV for location purposes. Function KROSS assists in locating the time at which specified state conditions are met. NFIND is used to identify the first cell of an entry having an attribute with a specified value in a particular file.

2.3.4.2.4.1 Function KROSS. This function locates specified state conditions and returns a coded value indicating whether the conditions have been met, not met, or exceeded.

Function KROSS is used primarily in subroutine SCOND and performs the dual functions of causing subroutine GASP (through its calls to SCOND) to "search" for specified state conditions and marking the satisfaction of those conditions. The first function is accomplished through use of the GASP IV control variable ISEES. The second is accomplished by returning a coded value of KROSS, indicating that a crossing has occurred.

A crossing may be either positive or negative. A positive crossing occurs when a variable (the crossing variable) increases in value from "less than" to "greater than" or "equal to" a second variable (the crossed variable), times a multiplicative constant, plus an additive constant. Both the crossing and crossed variables must be state variables when using KROSS, and they are identified by their subscripts. A zero value for the subscript of the crossed variable or a zero value of the multiplicative constant reduces the crossing description to one of a variable crossing a constant. The concept of a negative crossing is analogous to that of a positive crossing.

A crossing is defined as being located with sufficient precision if (1) a discrete change caused the crossing, in which case the crossing is located exactly in time or (2) a time advance caused the crossing and the difference at the end of the step between the crossing variable and the crossed function is less than, or equal to, the prescribed tolerance.

The user specifies not only the state conditions and tolerances defining a crossing but also the direction of crossing. Thus it is possible to search for and identify either negative or positive crossings or both. This flexibility permits, for example, the location of a local

maximum (or point of inflection) by defining a negative crossing of zero by the derivative of the variable of interest.

2.3.4.2.4.2 Function NFIND. Function NFIND is used to locate entries in NSET. NFIND is set equal to the first cell number of the entry. The first argument specifies a value for attribute comparison. The second argument of NFIND is the code for the type of search. Argument 3 specifies the file number, and argument 4 specifies which number of the attribute is compared. The fifth argument is a tolerance for searches seeking equality.

2.3.4.2.5 Data collection, computation, and reporting

GASP IV includes eight subroutines that support data collection, computation, and reporting. Subroutines COLCT, TIMST, HISTO, and GPLOT each perform all three functions. The user normally employs only the collection mode since the computation and reporting mode are automatically used during the preparation of a GASP IV summary report. Subroutine TIMSA is used only for data collection. Subroutines PRNTQ and PRNTS perform only computation and reporting. They are normally used during the preparation of the echo check and again during the preparation of the summary report. Subroutine SUMRY performs only a reporting function. It is normally called by subroutine GASP to provide a complete summary report. However, SUMRY can be called by the user at any time without altering any GASP IV variables.

2.3.4.2.5.1 Subroutine COLCT. This subroutine collects sample data for variables based on an observation of the variable when used in the collection mode. In the computation and reporting mode, it calculates the mean, standard deviation, standard deviation of the mean (assuming independent observations), coefficient of variation, minimum, maximum, and number of observations. It also provides tabular output of the calculated statistics.

2.3.4.2.5.2 Subroutine TIMST. Subroutine TIMST is also used for collecting values of variables for which statistical estimates are desired. When TIMST is used, the variable in question is assumed to have maintained a constant value over a time interval. This type of variable is referred to as a time persistent variable. An example of a time persistent variable would be the number of customers in a queueing system. The number of customers in a queueing system has a constant value over a time interval before it is changed. The length of the time interval dictates the weight assigned to the value of the variable in computing its

average over the entire simulation. Subroutine TIMST integrates the value of the variable over the time interval by multiplying these two quantities together.

2.3.4.2.5.3 Subroutine TIMSA. Subroutine TIMSA performs the same function as subroutine TIMST except that the average of the last value observed and the current value are used to integrate the time persistent variable. TIMSA is particularly useful for keeping statistics on state variables, since their values are permitted to change between events as well as at event times. The trapizoidal integration used in TIMSA yields better estimates for such variables than does TIMST. By data input, the number of variables for which TIMST and TIMSA are used is defined as the variable NNSTA. Clearly, since SSTPV is used by TIMST and TIMSA, different numeric codes must be assigned to variables using these two subroutines. Initial values of time persistent variables for statistical calculations must be set either by standard GASP IV input, by an initial call to TIMST or TIMSA, or by setting SSTPV(I,6) in the main program.

2.3.4.2.5.4 Subroutine HISTO. Subroutine HISTO is used to determine the relative frequency with which a variable falls within a set of prescribed limits. It is normally used for variables that have a prescribed value based on an observation, such as the time in the system for a customer, TISYS. The lower limit and width of each cell of the histogram are described by data input for each histogram. The array JJCEL is used to store the histogram. The number of cells for histogram I is specified by input and stored in the array NNCEL(I). The upper limit of the first cell and the width of each cell of histogram I is specified by input and stored in the arrays HHLOW(I) and HHWID(I), respectively. As with subroutines COLCT and TIMST, subroutine HISTO is used to print and to plot histograms and can be called by the user. Histograms are automatically printed at the end of a simulation run as part of the GASP IV summary report.

2.3.4.2.5.5 Subroutine GPLOT. Subroutine GPLOT collects values for eventual plotting of up to 10 dependent variables for one independent variable. The independent variable must be monotonic. Many options are available with regard to the type and scaling for the plot. At most, 10 plots, each with 10 dependent variables, can be stored on tape units. Significant reduction in computing time is obtained when the plot data are stored in core memory. GASP IV provides the capability for maintaining the values for one plot in core.

The first argument of GPLOT is a dimension variable that accepts up to 10 values. Since the dummy argument in the subroutine is dimensioned, there are various ways of passing the dependent variables to the plotting subroutine.

2.3.4.2.5.6 Subroutines PRNTQ and PRNTS. Subroutines PRNTQ and PRNTS are used to print the filing array and the state storage vectors, respectively. The statement CALL PRNTQ(2) would cause the computation and printing of the average number, standard deviation, and maximum number of entries in file 2, and would prepare a tabular listing of the contents of file 2. The statement CALL PRNTQ(0) would perform the procedure for all files. The statement CALL PRNTS would cause the state variables and their derivatives to be printed out.

When the entire file storage area is requested, subroutine PRNTQ computes and prints the maximum number of entries stored in the file storage array since it was last initialized. This information is obtained from a sequential search of predecessor pointers. The first entry having a negative predecessor pointer is the first entry that has not been used. The normal listing for each file includes: (1) the current simulated time, (2) the time the file was last used, (3) the time-integrated average and standard deviation of the number of entries in the file (not included if current time is equal to file initialization time), (4) the maximum number of entries in the file since last initialization (not included if current time is equal to initialization time), and (5) a sequential listing of the entries currently in the file.

2.3.4.2.5.7 Subroutine SUMRY. SUMRY provides all standard GASP IV output by either printing or causing all of the following to be printed:

1. A heading,
2. All parameter sets,
3. All statistics for variables based on observation,
4. All statistics for time persistent variables,
5. All files and file statistics,
6. All state and derivative values,
7. All histograms, and
8. All tables and plots.

2.3.4.2.6 Program monitoring and error reporting

The functions of monitoring and error reporting are supported by subroutines MONTR and ERROR. These subroutines are good candidates for user modification to obtain specific information for use during the debugging of a simulation program.

2.3.4.2.6.1 Subroutine MONTR. This subroutine selectively provides any of the following functions during a simulation run:

0. Start or stop tracing each event.
1. Print contents of files.
2. Reset file statistics and clear statistical storage arrays.
3. Print contents of state storage arrays.
4. Print information on state variables and files.
5. Print summary report.
6. Cause error exit.

Subroutine MONTR may be called directly by the user (after assigning an appropriate value to the event code, JEVNT), or by the occurrence of a monitor event (any time-event with an event code less than zero).

2.3.4.2.6.2 Subroutine ERROR. ERROR is called when an error is detected in a GASP IV subprogram. The error message provides the programmer with a code (KODE) that indicates the condition causing the error and the location of its detection.

Because ERROR may be called by subroutines that are called by ERROR, a loop protection variable (NERR) is used to assure that a loop does not persist.

Subroutine ERROR provides useful diagnostic information by listing the error code, simulated time, and current values in the file storage buffer. Next, a call to the user-provided subroutine UERR permits any specific information to be output. A complete summary report is then prepared (if possible) and the error message is repeated. Finally, ERROR causes a FORTRAN error to take advantage of FORTRAN-provided diagnostic and traceback information.

2.3.4.2.7 Miscellaneous support

The miscellaneous support subroutines included with GASP IV are functions SUMQ, PRODQ, and GTABL and subroutine GDLAY.

2.3.4.2.7.1 Functions SUMQ and PRODQ. The function SUMQ computes the sum of all attribute values for a given file. Suppose that the assets of a corporation are stored in file 2 with attribute 3 being used to define the value of an asset. The statement TASSC = SUMQ (3,2) computes the value of the total assets of the corporation as TASSC. The function PRODQ performs a similar function to SUMQ but multiplies the values of the attributes instead of summing them. Therefore, if the components logically, in series, in a system, are stored as entries in file 3, and attribute 4 is the reliability of a component, the statement SREL = PRODQ(4,3) will compute the system reliability SREL.

2.3.4.2.7.2 Function GTABL. The function GTABL provides a look-up function for a table in which the independent variable of the table has equal intervals. The arguments of GTABL include: (1) the name of the array storing the table that defines the function, (2) the value of the independent variable for which the function is to be evaluated, (3) the value of the independent variable corresponding to the first and last tabulated value, (4) and the interval of the independent variable corresponding to the interval between the tabulated values.

2.3.4.2.7.3 Subroutine GDLAY. Subroutine GDLAY provides a vehicle to obtain a variable order exponential delay for use in systems dynamics and simulations. Using this subroutine, state variables can be driven from their current value to a prescribed value in a specified number of stages. At each stage a specified amount of delay is encountered. To use GDLAY, each stage must be represented by a DD(·) variable. GASP IV automatically updates corresponding SS(·) variables in subroutine GASP.

2.3.4.2.8 Dummy subroutines

Dummy subroutines for all the user written subroutines are included in the GASP IV package. Their inclusion permits the user to code only those subroutines that are needed for a given simulation. The executable statement in each of the dummy subroutines is to comply with the requirement of ANSI FORTRAN that every subprogram have at least one executable statement.

2.3.4.2.9 Random deviate generators

To write general purpose subprograms for generating random deviates (samples), one must be able to reference parameter values for the distributions to be sampled within various subprograms. This is accomplished in GASP IV through a two dimensional array, PPARM. Each row of PPARM contains a set of values that are used as the parameters of a distribution. Since the parameters are stored in rows of PPARM, it is only necessary to designate the row number when calling a subprogram that requires parameter values. PPARM is initialized in subroutine DATIN through data cards, and the values of PPARM are printed to ensure that the values used in a simulation are recorded.

GASP IV provides the mechanism to obtain deviates given a distribution type and parameters for the distribution. The data collection, statistical analysis (including goodness-of-fit testing), and modeling to describe the inputs to a simulation must be performed by the user.

GASP IV provides subprograms for generating deviates from the following distribution types as listed in Table 2.3.4.2-2: uniform or rectangular, triangular, normal, log-normal, Erlang (gamma with an integer parameter), Gamma, Beta, and Poisson. The uniform, triangular, Erlang, and Poisson types employ the probability-integral-transformation approach to random deviate generation. The Beta and Gamma generators use a derived analytic result in conjunction with a rejection technique. A deviate from the exponential distribution can be obtained from the Erlang function or through the writing of a single statement.

2.3.5 User-Written Subroutines

2.3.5.1 Subroutine Descriptions

In addition to the main program, the user communicates with GASP IV through the subroutines described in Table 2.3.5.1-1.

The function of the main program, EVNTS, INTLC, and OTPUT were described previously. Subroutine UERR allows the user the option of performing operations and printing specific information when a GASP IV error is detected.

TABLE 2.3.4.2-2. RANDOM DEVIATE GENERATORS

Random Deviate Generator	Type of Distribution
DRAND	Uniform distribution on interval 0-1 (pseudorandom number).
UNFRM	Uniform distribution on a specified interval (ULO, UHI).
TRIAG	Triangular distribution.
RNORM	Uniform distribution on the specified portion of an array.
RLOG	Lognormal distribution.
ERLNG	Erland distriubtion (gamma distribution with an α parameter).
NPSSN	Poisson distribution.
GAM	Support routine used to obtain a sample from a Gamma or or Beta distribution.
GAMA	Gamma distribution.
BETA	Beta distribution.

2.3.5.2 Input Formats

All GASP IV variables are initialized by data card inputs. Each data input is associated with a specific data card in a specified format. A total of fourteen data card types are specified for complete initialization.

TABLE 2.3.5.1-1. DESCRIPTION OF USER-WRITTEN SUBROUTINES*

Name	Description
Subroutine EVNTS (IX)	Called by subroutine GASP to process event IX. A computed GO TO statement with argument IX is used to transfer to the appropriate time-event processing subroutine; however, if IX = IIEVT must be made by the user in EVNTS; normally the vector LFLAG is used for this purpose where LFLAG(I) is set by the user in subroutine SCOND; the form of subroutine EVNTS is illustrated in Figure 3-3.
Subroutine OTPUT	OTPUT provides a way for the user to obtain output in addition to the standard GASP IV summary report; OTPUT is called prior to subroutine SUMRY and can be used as an end-of-simulation event.
Subroutine UERR (KODE)	Called by subroutine ERROR to allow the user to print specific information if an error occurs.
Subroutine INTLC	Can be used to initialize state variables and non-GASP variables; called in subroutine DATIN after all GASP IV data cards have been read.
Subroutine STATE	Defines state variables through a listing of difference or derivative equations or both; called by subroutine DATIN (after INTLC is called) and then by subroutine GASP to compute current values of state variables; TNOW is the time to which GASP IV is trying to advance when STATE is called; DTNOW is the increment involved in the current update.
Subroutine SCOND	Called by subroutine GASP to determine if a state-event has occurred. State-events are normally defined in terms of state variables crossing a prescribed threshold or a prescribed variable with a tolerance specified for the amount of overcrossing. In SCOND, the user must make certain the value of the GASP IV variable ISEES is set to communicate if a state-event has been passed (ISEES < 0), if no state-event has been passed (ISEES = 0), or if no state-event has been passed but one does end the current step (ISEES > 0). When function KROSS is used, ISEES is automatically set; the user should indicate through the setting of a variable when a state-event occurs; the GASP IV vector LFLAG can be used for this purpose and the user can test LFLAG in the user written state-event routines; the LFLAG vector is not tested in GASP IV; after a return from EVNTS, subroutin GASP sets LFLAG(I) = 0 for I = 1, NFLAG.
Subroutine SSAVE	Called by subroutine GASP normally to record system status; the value of DTSAV specified the frequency with which SSAVE is called as follows: DTSAV < 0 SSAVE called only at event times DTSAV = 0 SSAVE called at each accepted update point, that is, at the end of each time step (which includes all event times) DTSAV > 0 SSAVE called each DTSAV time units and at event times Through SSAVE the user collects the data desired for eventual printout.

* The user normally also writes a subroutine for each type of event included in a model.

2.3.6 SIMNUC

2.3.6.1 Purpose of Computer Program

The Basic Simulator (SIMNUC) is an integrated package of subprograms designed to facilitate modeling and simulation of discrete stochastic systems in a manner similar to the GASP IV simulation programs.

The following features characterize this package:

1. Model independence.
2. FORTRAN orientation - the user's portion of a simulator can be programmed in FORTRAN or, if desired, in assembly language.
3. Capability to produce event-oriented simulation models.
4. Availability of list processing and dynamic memory management facilities.
5. Capability to collect and display standard queue and sample statistics.
6. A full complement of random number generators.

The basic approach, which sometimes is referred to as a *simulation-world-view*, used to model discrete systems for digital simulation with the Basic Simulator is the *event-oriented* approach, which emphasizes decomposition of the simulation process into individual event procedures, each of which describes all changes in the system caused by the occurrence of the related event, just as was done in GASP IV.

2.3.6.2 Simulation Facilities

The Basic Simulator consists of the following functional software components:

1. Simulation Run Control,
2. Dynamic Memory Management,
3. List Processing,
4. Random Number Generators,
5. Sample Statistics Processing, and
6. Error Diagnosis and Reporting.

2.3.6.3 Program Organization and Simulation Run Control

The components of the Basic Simulator are designed as independent FORTRAN callable

subprograms so that the user need only include those subprograms necessary for his particular simulation. The Basic Simulator subprograms may, in turn, require other subprograms such as diagnostic routines. The only case with which the user must be concerned is the Dynamic Memory Component, which is initialized if the List Processing or Simulation Control Components are used. By including the block data subprogram, SSDATX, all components except Dynamic Memory are initialized. Dynamic Memory is initialized by a call to the subprogram MINITX.

As indicated in Figure 2.3.6-1, the Main Program consists of two linearly structured program segments, which will be called the *Initialization* and *Termination Parts* of the *Main Program*.

The main function of the Initialization Part is to complete the following task:

1. Initialize a relatively small set of model-independent, Basic Simulator control parameters, such as those used by the Simulation Control and Dynamic Memory Components (this task is further discussed in "Special Programming Requirements").
2. Read input data and possibly print its summary; the input data typically can be divided into two categories: (a) the user-supplied simulation control parameters, and (b) model definition data.

After reading all input data, the internal representation of model definition can be completed—this action may require processing of the model definition data and construction of various data structures (such as tables, lists, etc.) from the processed data.

3. Initialize the problem-dependent control parameters and data structures in order to bring the system to be simulated to its initial state; this task may include (a) creation, or definition, of sample statistic-gathering mechanisms, (b) definition of model dependent queues, (c) initialization of random number sequences.
4. Define all event coordination structures.
5. Schedule initial events.

The Dynamic Memory is initialized by calling the subprogram MINITX. Since MINITX builds the Free List, it must be called before any other routine that assesses Dynamic Memory subroutines, either directly or indirectly, is called.

The user must properly define a labeled COMMON block, MEMXXX, containing the

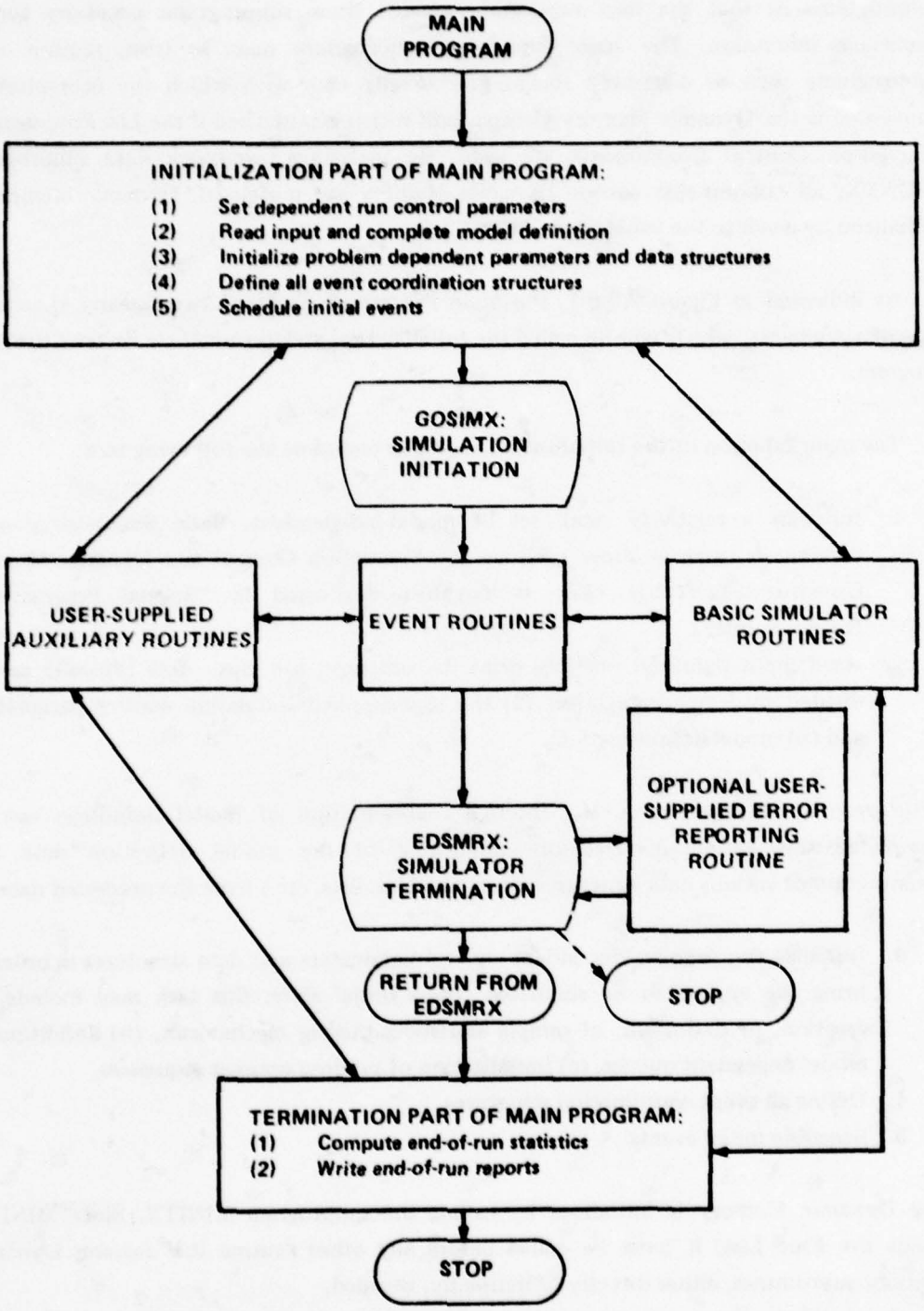


Figure 2.3.6-1. Interface and structure of the main program.

MEMORY vector for Dynamic Memory storage space; the user must also specify all required equivalences and types for the variables used in setting up the Dynamic Memory framework.

In the initialization part of the main program, a separate event coordination structure must be defined by calling DEFCSX for each class of blockable events in the simulation model.

After all initialization is complete, the simulation process is started by calling the subprogram GOSIMX. This FORTRAN subroutine executes the simulation run control functions which can be stated as follows:

1. Transfer the run control from the initialization Part of the Main Program to the Simulation Process (i.e., events posted on the Future Event List).
2. Establish the first executable statement which follows the call to GOSIMX as the reentry point to the Main Program.

3. Post the event notice for the final event (for terminating the simulation process) to occur at a time point which is to be expressed in terms of the simulation time units.

The subprogram EDSMRX is responsible for determining the simulation termination conditions. Two types of simulation termination conditions will be recognized:

1. Normal termination of a simulation process, which technically means the occurrence of the *final event* that has been present by the user; and
2. Detection of an error condition.

After the run control has been transferred to the simulation process (i.e., to the events posted on the Future Event List), it is returned to the Main Program *only* after a termination condition of type one arises. Type 2 above refers only to those error conditions that can be detected by means of the Error Diagnosis and Reporting Subroutines; many other types of errors may arise during a simulation run, and they may remain undetected by these subroutines.

As indicated in Figure 2.3.6-1, GOSIMX is invoked from the Initialization Part of the Main Program (which means that it is a user-accessible subprogram), and EDSMRX is automatically called (i.e., without an explicit intervention by the user's part of the simulator) after one of the above-stated termination conditions arises.

If the termination is of type 2, the user-supplied error reporting routine is executed, if present, to provide the user with any partial results that he may require. The simulation program is then terminated without return to the Termination Part of the Main Program.

If the termination is normal (type 1), control is returned to the Termination Part of the Main Program which follows the call to GOSIMX. The statement immediately following the call to GOSIMX can be considered to be the starting point of the Termination Part of the Main Program.

As in the case of the Initialization Part, the functions of the Termination Part are model dependent and typically consist of the following tasks:

1. Compute the end-of-run simulation statistics.
2. Write end-of-run reports.

The computation of end-of-run statistics includes those operations on ample and queue statistics which can be performed only after the termination of the simulation process, such as obtaining final values for sample means and standard deviations. Certain standard types of sample and list (queue) statistics can be gathered during simulation by means of the facilities of the Sample Statistics Processing and the List Processing Components, respectively.

The Sample Statistics Processing Components also contain subroutines for performing the end-of-simulation computations on gathered statistics and for displaying computed results.

2.3.6.4 Component Functional Description

The subprograms of the Basic Simulator are divided into six major groups, called components, which perform the following functions: simulation control, memory management, list processing, random number generation, processing of sample statistics, and error diagnostics.

2.3.6.4.1 Control of the simulation process

This section describes the facilities in the Basic Simulator whose function is to provide the user with tools for initializing and controlling a simulation process. Collectively, these control facilities will be referred to as the Simulation Control Component (SCC) of the Basic Simulator.

The Simulation Control Component consists of programs to manage the orderly initiation of the simulation process, the transition from one event to another, and the termination of the simulation process. The Simulation Control Component of the Basic Simulator makes it possible to simulate concurrent processes by representing each process as a sequence of events. In a simulation program, events are represented by the so-called event routines. The occurrence of an event during simulation is represented by a data structure, called an event notice, which contains the following information about the event:

1. The future time-of-occurrence of that event,
2. The entry point address of the corresponding event routine, and
3. Additional information, if any, to be passed to the event routine.

The last item provides the capability to pass input parameters to event routines.

2.3.6.4.1.1 Functional Description. Since control of the Basic Simulator is exercised by following the next-event principle, as in GASP IV, the user's portion of the simulator software consists of a master (main) program and of the so-called event routines, each of which represents an event type in the dynamic model of the system under consideration.

An event routine must be coded as a FORTRAN subprogram without arguments. Parameters are passed to it by means of the event notice, or by common blocks. Each event routine is responsible for the disposition of the event notice which invoked it. It may either reschedule the event notice with the subprogram SCHDEX or free the dynamic memory block containing the event notice. The event routine is also responsible for scheduling any successor events. A RETURN statement must always be used to exit from an event routine. This causes the transfer of control to the event routine which is to be executed next. The user's portion of the simulator may contain additional subroutines which are invoked from the main program or from the event routines.

A simulation run starts by transferring the control to the initial segment of the main program whose function is to perform various tasks associated with the initialization of simulation. Defining the lists to be used in simulation, setting up mechanisms for statistics collection, or reading and preprocessing of model data are examples of initialization tasks. Once this has been done the run control must be passed to the Simulation Clock Manager. The latter is a mechanism, consisting of several subroutines of the Basic Simulator that are not explicitly accessible to the user, whose main function is to manage the Future Event List, a major data structure used by the Simulation Clock Manager. The functions performed by the Simulation Clock Manager subprograms are to:

1. Retrieve the notice of the most imminent future event which now becomes the *current event*);
2. Advance the simulation clock to the time value indicated on the retrieved event notice;
3. Transfer the run control to the *event routine* which handles all events of the type to which the current event belongs;
4. Generate and post future event notices during the execution of the event routine; and
5. After the execution of the event routine is completed, return the run control to the Simulation Clock Manager, which then recycles the steps (a) through (e).

The simple simulation control scheme described above is not sufficient for logically more complex modeling situations which, for example, may require:

1. Suspension of the execution of an event routine, if a test performed after entering it indicates that the logical conditions for its further execution do not yet exist; or
2. Cancellation of an event notice already posted on the Future Event List.

The Simulation Control Component contains facilities for handling these two above described situations. The first one is handled by means of the so-called event coordination structures which are lists used to store the notices of suspended events. Subprograms are provided for defining such a structure, for suspending an event by placing its notice on the appropriate event coordination structure, and for releasing all event notices currently on a specified event coordination structure. Similarly, a subprogram is provided for cancelling the notice on the Future Event List of a future failure event.

2.3.6.4.1.2 SSC SUBROUTINES. The Simulation Control Component of the Basic Simulator contains the following subroutines:

1. GOSIMX — Start the Simulation Process,
2. ENTRYX — Get the Entry Address to a Subprogram,
3. SCHDEX — Schedule an Event,
4. CALLEX — Get the Entry Address of the Calling Program,
5. DEFCSX — Define an Event Coordination Structure,
6. BLCKEX — Block an Event Notice,
7. RLSEEX — Release an Event Notice,
8. FINDEX — Get the Pointer to an Event Notice,
9. CANCLX — Cancel an Event Notice, and
10. TIMEXX — Read the Clock.

Initialization of the simulation process, which technically means transfer of control from the user's main program to the Simulation Clock Manager, is the function of GOSIMX, the first of the above ten subroutines. Hence, GOSIMX must be called from the user's main program after the completion of tasks associated with the initialization of simulation. Once the simulation control is passed to the Simulation Clock Manager, appropriate user-supplied event routines are called by the Simulation Clock Manager according to the event notices posted on the Future Event List until the end-of-simulation time is reached. At that instant, the Simulation Clock Manager returns the execution control to the user's main program, with the reentry point being the first statement following the call to GOSIMX.

The remaining subroutines, following GOSIMX in the above list, are normally called from the user-written event routines to perform various simulation control functions such as posting a future event notice, blocking and then releasing an event, cancelling a posted event notice, getting the current simulation time, or getting a pointer to an event routine.

2.3.6.4.1.3 GOSIMX – Start the Simulation Process. This subroutine triggers the simulation process by executing the transfer of control from the user's main program to the Simulation Clock Manager; it also sets up a mechanism for returning the control to the main program at the end of the simulation process. Control is returned to that statement in the main program which follows the call to GOSIMX. More specifically, GOSIMX performs the following tasks:

1. Transfer the program-run control from the initialization part of the main program to the Simulation Clock Manager (the latter may be thought of as being a simulation control mechanism which is not explicitly accessible to the user).
2. Establish the first executable statement which follows the call to GOSIMX as the reentry point to the main program.
3. Post the Event Notice for the final simulation event which will return control to the main program.
4. Specify the entry point of a user-supplied subprogram which is to be executed in the event of an error exit termination of the simulation process.

The call statement for GOSIMX contains two arguments. The first argument is used to specify the time of termination of the simulation. If a user-supplied subroutine is to be executed before the termination of the simulation process due to a detected error, the second argument must contain a pointer to that subroutine; otherwise, it must have zero value.

2.3.6.4.1.4 ENTRYX – Get the Entry Address to a Subprogram. Subprogram ENTRYX computes an entry point address of (pointer to) a FORTRAN subprogram.

2.3.6.4.1.5 SCHDEX – Schedule an Event. This subprogram places an event notice on the Future Event List. The arguments of SCHDEX contain user-supplied information, specify the time when the event should occur, and normally contain the entry point address of the event routine.

2.3.6.4.1.6 CALLEX – Get the Entry Address of the Calling Program. When invoked in a subprogram, CALLEX computes the entry point address of the program which called the subprogram.

2.3.6.4.1.7 DEFCSX – Define an Event Coordination Structure. This subprogram is used to define an event coordination structure (ECS). A separate event coordination structure must be defined in the initialization phase of a simulation run for each logically distinct class of blockable event notices in the simulation model. The arguments of DEFCSX include the name to be assigned to the ECS, and the pointer to be used for all future references to this ECS.

2.3.6.4.1.8 BLCKEX – Block an Event Notice. This subprogram is used by an event routine to block its own current execution by putting the event notice for itself on an appropriate event coordination structure. A blocked event remains on an event coordination structure until subroutine RLSEEX, described in the sequel, is called to release it. BLCKEX returns the execution control to the calling event routine which is blocking itself; hence, further execution of this routine must subsequently be halted by a RETURN statement. The arguments of BLCKEX include a pointer to the event coordination structure on which the notice is to be placed and a timing parameter used by RLSEEX in order to determine the time at which the released event is to be rescheduled.

2.3.6.4.1.9 RLSEX – Release an Event Notice. This subprogram is used to release all event notices that have been blocked on a specified event coordination structure. Technically, the release of blocked event notices means computing new future occurrence times for the blocked events, putting these time values in the event notices, and then moving these notices to the Future Event List.

2.3.6.4.1.10 FINDEX – Get the Pointer to an Event Notice. This subprogram searches for a specified event notice on the Future Event List (FEL). If FINDEX succeeds in locating this notice, it computes and returns a pointer to the notice; otherwise, this pointer is given zero value.

2.3.6.4.1.11 CANCLX – Cancel an Event Notice. This subprogram is used to cancel a specified event notice on the Future Event List. The event notice is specified by giving its pointer, which may be computed by using FINDEX or obtained by some other means.

2.3.6.4.1.12 TIMEXX – Read the Clock. This program computes the current value of the simulation clock (time), which is a standard-length integer.

2.3.6.4.2 Memory management

The Dynamic Memory Management Component of the Basic Simulator provides the user with the capability to manage dynamically during the program execution a vector of COMMON memory so that blocks from the vector can be temporarily allocated to the user's program and then deallocated (freed) when no longer needed. This memory management method makes it possible to reuse the same storage many times for different purposes throughout a simulation run, and thus not only to minimize total program storage requirements, but also to increase the size of the simulation problem that can be handled. With this method of memory management, the following allocation is used:

1. A block in dynamic memory must always be allocated (deallocated) by appropriate action in the program, and
2. An allocated block can be accessed only through a pointer which is set up at allocation time.

2.3.6.4.2.1 Functional Description. The Dynamic Memory Management Component allocates storage from a vector called MEMORY which is contained in the labeled COMMON block MEMXXX. The user's program accesses a block allocated from Dynamic Memory through a pointer associated with that block. At any instant of program execution, the vector MEMORY consists of two types of blocks: those that are allocated and currently in use, and those that are available for use. All deallocated blocks are chained together into a list called the Free List. It is the user's responsibility to put memory deallocation requests in appropriate places in his program.

When the program requests a block of certain size, the Free List is searched to find a deallocated block of sufficient size. The search/allocation strategy works as follows: during the search, physically adjacent blocks in the Free List are coalesced into a single large block; the requested block is allocated from the first sufficiently large free block encountered in this search. If no such block can be found, further execution of the simulation program is terminated.

2.3.6.4.2.2 DMMC SUBROUTINES. The Dynamic Memory Management Component consists of the following subroutines:

1. MINITX — Initialize Memory,
2. MEMALX — Allocate Memory,
3. MEMFRX — Free (Deallocate) Memory,
4. MEMZOX — Store Zeroes in Memory,
5. MEMDPX — Dump Memory,
6. MCOPYX — Copy Memory, and
7. MSTATX — Write Memory Statistics.

To use these subroutines, the user must properly define a labeled COMMON block, MEMXXX, containing the MEMORY vector for Dynamic Memory storage space; the user must also specify all required equivalences and types for the variables used in setting up the Dynamic Memory framework.

2.3.6.4.2.3 MINITX — Initialize Memory. MINITX is used to initialize the memory vector as the Free List from which blocks of memory are to be allocated. The arguments of MINITX are the length in words of the memory vector specified by the user and the length in words of the desired block modulus.

2.3.6.4.2.4 MEMALX — Allocate Memory. This subroutine is used to allocate a block of memory from the Free List. The user's portion of an allocated block is automatically zeroed. If such a block cannot be allocated due to the lack of storage, simulation is terminated.

2.3.6.4.2.5 MEMFRX — Free (Deallocate) Memory. MEMFRX is used to free an allocated block of dynamic memory in order to return it to the Free List.

2.3.6.4.2.6 MEMZOX — Store Zeroes in Memory. The subroutine MEMZOX can be used to store zeroes in all user-available words of a specified Dynamic Memory block.

2.3.6.4.2.7 MEMDPX — Dump Memory. The subroutine MEMDPX can be used to dump the contents of Dynamic Memory for program debugging purposes.

2.3.6.4.2.8 MCOPYX — Copy Memory. This subroutine can be used to create a copy of the contents of a block of Dynamic Memory and return a pointer to the block of Dynamic Memory containing the copy. The contents of the original block are unmodified.

2.3.6.4.2.9 MSTATX – Write Memory Statistics. The subroutine MSTATX writes the Dynamic Memory statistics for the line printer. These statistics are:

1. The size of the memory vector,
2. The maximum number of words which were allocated at any time until the call to MSTATX, and
3. The number of words currently allocated.

2.3.6.4.3 List processing

This section discusses list processing facilities available to the user. The List Processing Component (LPC) of the Basic Simulator provides tools for handling two types of list structures, doubly linked lists (queues) and indexed lists. In addition, the LPC contains subroutines for collecting standard list statistics.

2.3.6.4.3.1 Functional Description. All lists maintained by the List Processing Component use the facilities of Dynamic Memory both for storing list heads and list elements. For both types of lists, the list head is created and maintained exclusively by LPC subroutines. In both cases, the list to be processed is identified to a List Processing Component subroutine by means of a pointer to the head of that particular list (such pointers will be called list-head pointers).

To initialize (create, define) a list, the user's program must set aside a variable which is to function as the list-head pointer for that list; then appropriate action must be taken to create the list-head, which differs for each type of list.

2.3.6.4.3.2 LPC Subroutines. The list processing subroutines available to the user are listed below. For convenience, these subroutines are divided into two groups, those for processing doubly linked lists and those for handling indexed lists:

1. Subroutines for Processing Doubly Linked Lists:
   a. LTDEFX – Define a List,
   b. LTADDX – Add an Element to a List,
   c. LTDMPX – Dump a Linked List,
   d. LTFNDX – Find a Specified Element in a List,
   e. LTNXTX – Find Next (First) Element in a List, and
   f. LTRSPX – Remove a Specified Element From a List.

2. Subroutines for Processing Indexed Lists:
   a. DSADDX — Add a New Member to an Indexed List,
   b. DSDMPX — Dump an Indexed List,
   c. DSNXTX — Return a Pointer to a Specific Member of an Indexed List, and
   d. DSPRGX — Purge an Indexed List.

2.3.6.4.3.3 Processing of Doubly Linked Lists by Means of LPC Subroutines. Every element of a doubly linked list consists of two parts, those being a fixed-length part followed by a variable-length part. The fixed part of an element always consists of three words, which in the indicated order, contain the forward pointer, the backward pointer, and the time when the element was added to the list.

LTDEFX — Define a List. This subroutine creates a standard list head for the list being defined and then stores in it information which defines and initializes the new list. It returns a pointer to the newly created list head. The list-head pointer of a doubly linked list points to a table of standard format.

LTADDX — Add an Element to a List. This subroutine adds a new element (a block stored in Dynamic Memory and pointed to by the input argument *bkptr*) to the list identified by the list-head pointer *ltptr*. Addition of a new element is done by properly linking it with the elements already in the list and by modifying the list-head. The insertion position relative to other list elements is determined by means of the queueing discipline code of the list, which is located in the list-head. Thus, the code value:

1. LIFO causes the new element to be added to the logical front end of the list,
2. FIFO causes the new element to be added to the logical back end of the list,
3. LOHI causes the new element to be placed logically in front of the first element with a rank value larger than that of the new element, and
4. HILO causes the new element to be placed logically in front of the first element with a rank value smaller than that of the new element.

LTDMPX — Dump Contents of Linked List. This subroutine is used to write the contents of a Linked List according to a user-supplied format.

LTFNDX — Find a Specified Element in the List. This subroutine searches the list in the direction of the decreasing rank of its elements to find the first element which contains a specific value (not necessarily the rank) stored in the specified word of the list elements. On finding such an element, the subroutine returns a pointer, *elptr*, designating this element. If none is found, *elptr* is assigned a zero return value.

The pointer points to the list-head of the list to be searched. The type of the test value *data* must be the same as that of the tested quantity. The argument specifies the offset of the word in every list element whose value is to be compared with that of *data*. The element found by subroutine LTFNDX may be removed from the list by a call to subroutine LTRSPX.

LTNXTX – Return the Pointer to the Next (First) Element In a List. Given the pointer *elptr* to an element in a specified list, this subroutine returns the pointer to the element which is logically next (i.e., which is of the next lower rank) after the one pointed to by the *elptr*. If the user sets the input argument *elptr* to zero, then the routine returns the pointer to the first element of the list. If such an element (first or next) does not exist, then *nxtpt* has a return value of zero. The list is specified by its list-head pointer *ltptr*. The element found by subroutine LTNXTX may be removed from the list by calling subroutine LTRSPX.

LTRSPX – Remove a Specified Element From a List. This subroutine removes the element pointed to by the pointer *elptr* from a list specified by the list-head pointer *ltptr*. If *elptr* is set equal to zero, then the logically first element is removed. This subroutine can be used to remove elements found by subroutines LTFNDX or LTNXTX.

2.3.6.4.3.4 Processing of Indexed Lists by Means of LPC Subroutines. The list-head is a vector: the first component of this vector contains a counter which specifies the number of elements currently in the list; subsequent components of this vector contain the pointers to individual elements of the list. The user communicates with the indexed list through the list-head pointer.

Four subroutines for handling indexed lists are available. These subroutines are described next. Contrary to the case of doubly linked lists, there is no special subroutine for initializing an indexed list (i.e., for creating the list head for such a list); this is done by assigning the first element to the list to be created.

DSADDX – Add a New Element to an Indexed List. This subroutine adds a new element to an indexed list identified by the list-head pointer *lptr*. The element to be added must be a block stored in Dynamic Memory; the input pointer, *elptr*, must be pointing to the storage area of that element.

DSDMPX – Dump Contents of Indexed List. This subroutine is used to write the contents of an Indexed List according to a user supplied format.

DSNXTX – Return a Pointer to a Specified Element of an Indexed List. Subroutine DSNXTX retrieves the pointer *elptr* to the jth element of an existing (nonempty) indexed list which is identified by the list-head pointer *lptr*. The jth element is specified by assigning J as the value of the input argument, *index*, where $1 \leqslant J \leqslant N$ (where N is the number of elements currently in the list).

DSPRGX – Purge an Indexed List. This subroutine deallocates from Dynamic Memory the storage blocks occupied by the list-head and by all elements of the indexed list identified by the list-head pointer *lptr*. Then the subroutine sets the pointer *lptr* to zero. If the subroutine is entered with a list-head pointer value of zero, control is returned to the calling program.

2.3.6.4.4 Generation of random numbers

The Random Number Generation Component (RNGC) of the Basic Simulator provides the user with subroutines for producing sequences of random numbers from nine standard distributions commonly used in simulation. In addition, it contains three subprograms for generating random numbers from continuous or discrete distributions defined by the user.

2.3.6.4.4.1 Functional Description. Each RNGC subprogram—with the exception of the generator of random numbers having uniform distribution over the unit interval (subprogram RANDOX)—calls RANDOX to generate one or several uniformly distributed random numbers needed for computation of its own output. Thus, a sequence of random numbers from a nonuniform distribution always represents the result of no one sequence of random numbers from a uniform distribution. The first number is such a sequence; U_0 must either be furnished by the user or else it is automatically provided by the Basic Simulator during standard initialization of simulation.

To give the user some control over generation of the sequences of uniformly distributed random numbers which are to be used to generate random numbers from other distributions, a labeled COMMON block is made available to the user. Each component of the vector ISTART is associated with one and only one RNGC subprogram: initially, it must contain the seed for the underlying sequence of uniformly distributed random numbers; subsequently, it is used to store the generated successors of that seed. It is conceivable that the user may want to use the same generator subprogram to produce several sequences concurrently during simulation. This can be done by using an appropriate component of the vector ISTART to store the values associated with generation of each such sequence. To summarize the above ideas, the elements of vector ISTART contain the

initial (seed) and subsequent values that are used to propagate each needed sequence of uniformly distributed random integers. Some sequences of this type do not show up explicitly in simulation, for they are used as underlying sequences to produce random numbers from other distributions; the others are used, per se, in simulation.

The Random Number Generation Component consists of the following subprograms:

1. RANDOX — Random Number from Uniform Distribution Over the Unit Interval.
2. RMBINX — Random Number from a Negative Binominal Distribution.
3. RMCCPX — Random Number from a User-Defined Discrete Cumulative Distribution.
4. RMDCPX — Random Number from a User-Defined Discrete Cumulative Distribution.
5. RMDRWX — Random Number from a Bernoulli Trial.
6. RMERLX — Random Number from an Erlang (Gamma) Distribution.
7. RMEXPX — Random Number from an Exponential Distribution.
8. RMICPX — Random Integer from a User-Defined Discrete Cumulative Distribution.
9. RMIUFX — *Uniformly Distributed Random Integer from a Set of Consecutive* Integers.
10. RMNRLX — Random Number from a Normal Distribution.
11. RMPSNX — Random Number from a Poisson Distribution.
12. RMUFMX — Uniformly Distributed Random Number from a Specific Interval.

All these subprograms are FORTRAN FUNCTION subprograms. With a single exception, they return a single output quantity, which is either a single-precision floating point number or a standard-length integer. The only exception is RANDOX, which besides producing a floating point random number, also generates an integer-valued uniform random quantity. The use of each subprogram is discussed next.

2.3.6.4.4.2 RANDOX — Random Number from a Uniform Distribution Over the Unit Interval. Normally, RANDOX is used to generate a sequence of single-precision floating point values, $[U_n]$ which represent random numbers uniformly distributed over the unit interval [0,1]. The secondary use of RANDOX is to produce a sequence of random integers, $[I_n]$, uniformly distributed over an interval defined by the computer word size.

2.3.6.4.4.3 RMBINX — Random Number from a Negative Binominal Distribution. RMBINX computes a random number from the negative binominal distribution with parameters (P, N); i.e., given (1) the number N of successes in a sequence of independent

Bernoulli trials (i.e., each trial has only two outcomes—a success or a failure) and (2) the constant probability P of success in each trial, this subroutine generates the number of failures before the Nth success is observed.

2.3.6.4.4.4 RMCCPX – Random Numbers from a User-Defined Continuous Cumulative Distribution. Given the tabular approximation

$$(X(I), Y(I)) \text{ for } I = 1, 2, \ldots, N$$

to a cumulative probability distribution function $y = F(x)$ of a continuous random variable X, this subprogram computes a random number XR from the range $X(1) \leqslant X(N)$.

2.3.6.4.4.5 RMDCPX – Random Number from a User-Defined Cumulative Distribution. Given the tabular approximation

$$(X(I), Y(I)) \text{ for } I = 1, 2, \ldots, N$$

to a cumulative probability distribution function $y = F(x)$ of a discrete random variable X, this subprogram computes a random number XR from the set of values [X(1),X(2), . . . ,X(N)].

2.3.6.4.4.6 RMDRWX – Random Number from a Bernoulli Trial. A Bernoulli trial represents an idealized experiment which can have only two observable outcomes, such as a success and a failure, to be represented here by 1 and 0, respectively. Denote the probability of a success in a trial by P; then the probability of a failure is $1 - P$. Given the probability P of success, this subprogram returns 1 (= success) with probability P and 0 (= failure) with probability $1 - P$. The value of the input argument P must be in the range $0 \leqslant P \leqslant 1$.

2.3.6.4.4.7 RMERLX – Random Number from an Erlang (GAMMA) Distribution. This subprogram produces a random number XR from the Erlang (A, N) (or equivalently from the gamma (A, N)) distribution. Here, the positive real number A is the mean of the distribution; the positive integer N specifies the number of independent exponential random variables, each from the exponential distribution with the mean A/N, whose sum represents a random variable from the Erlang (A,N) distribution. Thus, a random variable XR from this Erlang distribution has expected value $E(XR) = A$ and variance $Var(XR) = A^2/N$.

2.3.6.4.4.8 RMEXPX – Random Number from Exponential Distribution. This subprogram computes a random number from the exponential distribution with parameter A. If

XR represents such a random number, then the expected value E(XR) = A and the variance Var(XR) = A^2.

2.3.6.4.4.9 RMICPX – Random Integer from a User-Defined Discrete Cumulative Distribution. This subprogram computes a random integer IR from a discrete cumulative distribution defined by the user over a set of consecutive integers. Such a distribution can be represented by a set of ordered pairs.

2.3.6.4.4.10 RMIUFX – Unfiformly Distributed Random Integer from a Set of Consecutive Integers. Consider an integer valued random variable X which is uniformly distributed over the set

$$IA, IA + 1 \ldots, IB$$

of consecutive integers; that is, if IB − IA = N − 1, then

$$P(X = x) = 1/N \text{ for } x = IA, IA + 1, \ldots, IB$$
$$P(X = x) = 0 \text{ for all other values of } x.$$

Given the values of IA and IB, this subprogram selects a random integer IR from one of the N equally probable values, IA, IA +1, . . . , or IB.

2.3.6.4.4.11 RMNRLX – Random Number from a Normal Distribution. This subprogram computes a random number from the normal distribution with the mean value A and with standard deviation B.

2.3.6.4.4.12 RMPSNX – Random Number from a Poisson Distribution. This subprogram generates a random number X from the Poisson distribution with the mean value A.

2.3.6.4.4.13 RMUFMX – Uniformly Distributed Random Number from a Specified Interval. This subprogram computes a random number from the uniform distribution over the interval [L, R].

The values of L and R must satisfy the inequality $L \leq R$.

2.3.6.4.5 Processing of sample statistics

In simulation of discrete-stochastic systems, a substantial portion of the output observed

by the simulator is collectable and expressible in the form of standard sample statistics, such as sample means, standard deviations, extreme values, or frequency distribution tables. This section describes the subprograms, collectively known as the Sample Statistics Processing Component (SSPC) of the Basic Simulator, whose function is to collect, process, and display, when told to do so, standard sample statistics for specified observable phenomena.

2.3.6.4.5.1 Functional Description. The sample statistics which can be obtained during simulation with the assistance of SSPC subroutines for an observable phenomenon consists of summary data and of an optional frequency distribution table. The latter can be built up in several alternative modes. The statistical summary data contain sample size, mean, standard deviation, and range (i.e., its minimum and maximum values). The frequency distribution table shows how sample values are distributed over a specified set of continuous intervals.

As stated below in the description of SSPC subroutines, a frequency table can be built up in one of the three possible modes, the difference, normal, or time-weighted mode. Furthermore, an entry to be added to a frequency table may optionally be multiplied by a weight coefficient. The latter option is available only for the first two modes of frequency table construction, the difference or the normal mode.

In the *normal mode* sampling for a frequency distribution table, the sampled value is first weighted by a multiplicative coefficient (if required—otherwise, the value of multiplicative coefficient can be thought of as being equal to one) and then added to the proper interval of the table.

If the *difference mode* is used to build up a frequency distribution table, the entry added is the weighted or nonweighted difference (depending on the weighting option) between the current sample value and that observed the last time. This sampling mode can be used to study distribution changes or rates of changes in a recurring phenomenon. Sampling in the *time-weighted mode* is performed as follows. The sample value to be added is first multiplied by the amount of time which elapsed since the last sampling instance for the frequency distribution table under consideration; next, this time-weighted value is added to the table as in the normal mode. Thus, the time-weighted mode can be used to study phenomena such as the distribution of storage occupancy.

Generating and displaying a frequency distribution table requires three functions. First, the table and the other sample statistics must be defined and initialized at the beginning of

simulation. Next, each time a quantity to be added to the table is generated in the simulation process, its value must be examined to find the appropriate position (range segment) of the table to which this value belongs and then the frequency of the position must be incremented. Similarly, one must update the quantities which are being generated to compute the sample mean, standard deviation, and extreme (minimum and maximum) values. Finally, at the end of simulation, computation of the above-mentioned statistics must be completed and then the results must be displayed by writing a computer-printed report. The Sample Statistics Processing Component provides three subroutines, described below, to accomplish these three functions.

2.3.6.4.5.2 SSPC Subroutines. The Sample Statistics Processing Component consists of the following three subroutines:

1. TBDEFX — Define/initialize the sample statistics,
2. TBDATX — Update the sample statistics, and
3. TBOUTX — Complete and output the sample statistics.

At the beginning of a simulation program, the user must call TBDEFX to initialize the statistics collection for each sample in which he is interested. Then, as sample observations are generated during a simulation run, the subroutine TBDATX is repeatedly called to accumulate and update sample statistics. At the end of the run, TBOUTX is called for each sample under consideration to complete computation of sample statistics and then to output the results. The use of each subroutine is discussed next.

2.3.6.4.6 Error diagnosis and reporting

The objective of the Error Diagnosis and Reporting Component (EDRC) of the Basic Simulator is to provide the user with tools for debugging his simulation program. This component consists of three basic types of subroutines:

1. Those for providing execution traces and for writing diagnostic messages.
2. Those for checking operational parameters of the Basic Simulator.
3. Those for printing the contents of data structures used in the simulation.

The subroutines of type two are not directly accessible to the user but are selected by a call to the debug control routine (or by default).

The error diagnosis and reporting routines are described below.

EDRC Subroutines:

2.3.6.4.6.1 DEBUGX – Select Debug Control. This subroutine is used to select the level of parameter checking and execution trace in the Basic Simulator. It may be called as often as desired during the simulation. If it is never called, default values are used.

2.3.6.4.6.2 DIAGNX – Write a Diagnostic Message. This subroutine is used to write a diagnostic message and to optionally terminate further execution.

2.3.6.4.6.3 TRACEX – Write a User Trace. This subroutine is used to write a user trace message containing the name of the calling routine, the simulation time, and the value of an input quantity.

2.3.6.4.6.4 DSDMPX – Write Contents of Indexed List. This subroutine is used to write the contents of an Indexed List according to a user supplied format.

2.3.6.4.6.5 LTDMPX – Write Contents of Linked List. This subroutine is used to write the contents of a Linked List according to a user supplied format.

2.3.6.4.6.6 MEMDPX – Dump Memory. The subroutine MEMDPX can be used to dump the contents of Dynamic Memory for program debugging purposes. Calling this subroutine has the following effect: The contents of all allocated blocks, including their block size, will be printed. The block size and pointer fields will be printed for all blocks in the Free List. The contents of those blocks that cannot be identified as clearly being of one of the two types (i.e., either allocated or else free) will be printed in conjunction with error recovery attempts. Each word will be printed both as a standard-length integer and in machine format.

2.3.6.4.6.7 MSTATX – Write Memory Statistics. The subroutine MSTATX writes the Dynamic Memory statistics for the line printer. These statistics are:

1. The size of the memory vector,
2. The maximum number of words which were allocated at any time until the call to MSTATX, and
3. The number of words currently allocated.

2.3.6.4.6.8 PKTPRX – Write Contents of Dynamic Memory Block. This subroutine is used to write the contents of a block of dynamic memory. The block is printed both as a standard-length integer and in machine format.

2.3.6.4.6.9 PKTWRX – Write Contents of Dynamic Memory Block. This subroutine is used to write the contents of a block of dynamic memory according to a user supplied format.

2.3.6.4.6.10 RPWRTX – Write Data. This subroutine is used to write any contiguous data according to a user supplied format.

SECTION 2.3 BIBLIOGRAPHY

GASP IV

Green, T. "GASP IV Listing of Tape Directory, GASP Source, DPM Source, SNK Subroutines, DAISIM Source," Unpublished AFAL document, 1977.

Pritsker, A., and Alan, B. "The GASP IV Simulation Language," John Wiley & Sons, Inc., New York, 1974.

SIMNUC

Texas Instruments, Inc., *Users Manual for Basic Simulator Computer Program*, Dallas, Texas, no date.

2.4 DISTRIBUTED PROCESSOR/MEMORY SYSTEM NETWORK SIMULATION SYSTEM

2.4.1 Introduction

The advent of the minicomputer, and subsequently the microprocessor and microcomputer, has placed a great deal of emphasis on the utilization of multiple computing elements in avionics systems. The relatively small size of minicomputers and microcomputers allows them to perform specialized, localized tasks while providing, in many cases, added performance over more general purpose computing systems. Multiplexed data buses have further provided the opportunity for computing elements to share resources and to provide the redundancy that can extend system operations in critical situations.

These added capabilities, provided by what may be referred to as distributed systems, present the system designer with some added complexities as well. Questions regarding how to partition the avionics tasks among several processors, the selection of the architecture of the processor interconnections, the information transfer rates between processors, the coor-

dination of information transfer, and the security and reliability of the overall system are more difficult to answer because of the large number of potential system configurations. Further, the basic information system design must be chosen early in the design process if a top-down design approach is to be followed. It is therefore essential to be able to explore the basic parameters of a distributed processor architecture by simulation so that performance parameters for individual processors, sensors, and information display elements can be specified.

The Distributed Processor/Memory (DP/M) System Network Simulator (SNS) provides the necessary tool to explore some of the tradeoffs available to the designer of distributed systems. The SNS is a discrete event-oriented high level traffic simulator written in ANSI standard FORTRAN. The SNS is built around a nucleus of model-independent utility routines (SIMNUC) which are not simulators in themselves, but are used to create a simulator in conjunction with the avionics software task specifications and topological organization specifications of a given avionic system.

2.4.2 DP/M Hardware Architecture

The DP/M system concept is essentially the use of varying numbers of simple, homogeneous processor/memory elements (PE's) applicable to a wide range of avionics system processing problems. Architecturally, these PE's can be used as standalone uniprocessors or they can be configured in a distributed network as shown in Figure 2.4.2-1. Serial-time-division-multiplex (TDM) buses interconnect the network. Two levels of busing are provided: a Global bus can interconnect each PE in a system network and a Local bus can interconnect multiple PE's clustered together to perform a given function. This cluster of PE's is referred to as an Affinity Group (AG). Input/output (I/O) for a given P/E to an external device is via its local I/O interface unit. While other more complex bus structures might be postulated, the structure provided by DP/M allows investigation of a number of important avionic information processing and control structures.

Although it is not necessary to know the structure of the PE's in order to use the SNS, such knowledge is useful in order to appreciate the architectural variations possible with DP/M SNS and their application to real avionics situations. Four functional modules make up the DP/M PE as shown in Figure 2.4.2-1. The Bus Interface Unit (BIU) is the basic TDM data transfer interface to the processor and memory. The BIU translates bit serial data into parallel words and transfers data and status information to the PE processor and memory. The processor module is the instruction-sequencing and data-processing portion of the PE. The memory module provides necessary program instruction and data storage, and can be

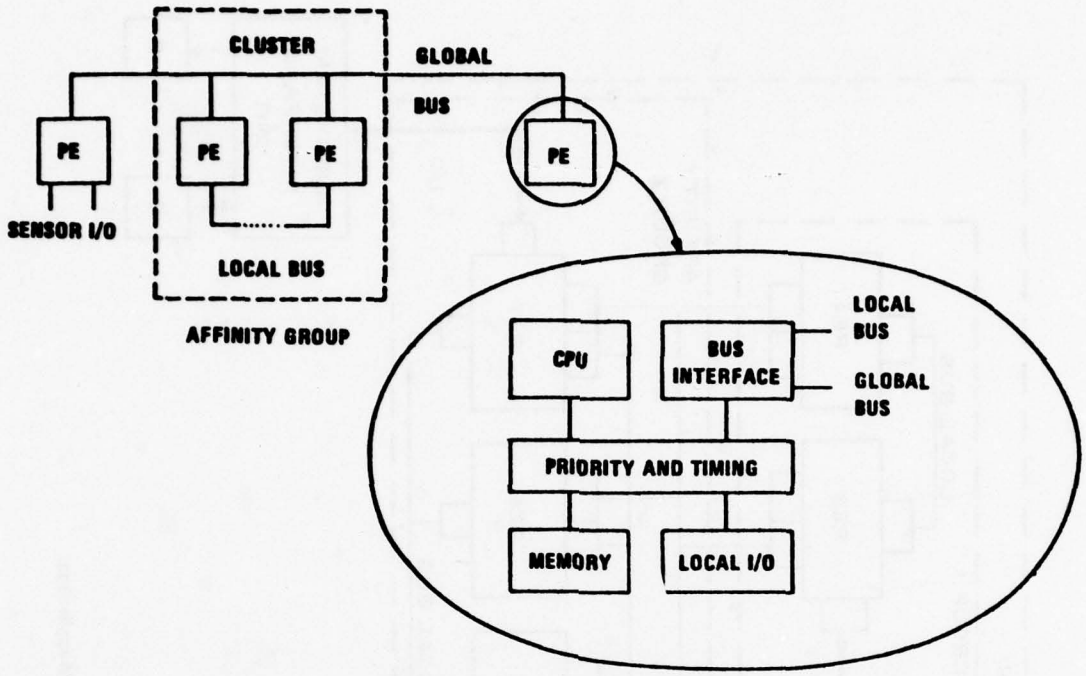


Figure 2.4.2-1. DP/M system architecture.

accessed by the other PE modules: the BIU, the processor, and the I/O interface. Memory access is local to a PE; i.e., access by another PE or shared memory is not part of the DP/M concept. The Input/Output Interface Unit (IOIU) of the PE permits digital data, command and status information transfers between the PE and the external devices in the avionics system.

An example of the type system envisioned for implementation with the DP/M architecture is shown in Figure 2.4.2-2. Note that the structure provides for a redundant Global bus. In this example, two Affinity Groups are formed by PE3-PE4 and PE5-PE6-PE7. Interface to sensors/effectors is illustrated via IOIU's in the PE's and subsystem interfaces external to the DP/M system.

2.4.3 Bus Control Protocols

2.4.3.1 Modified Round-Robin Message Broadcast

During the period the DP/M SNS has been in use at AFAL, two bus protocol algorithms have been implemented. The first is a modified "round-robin" slotting technique which

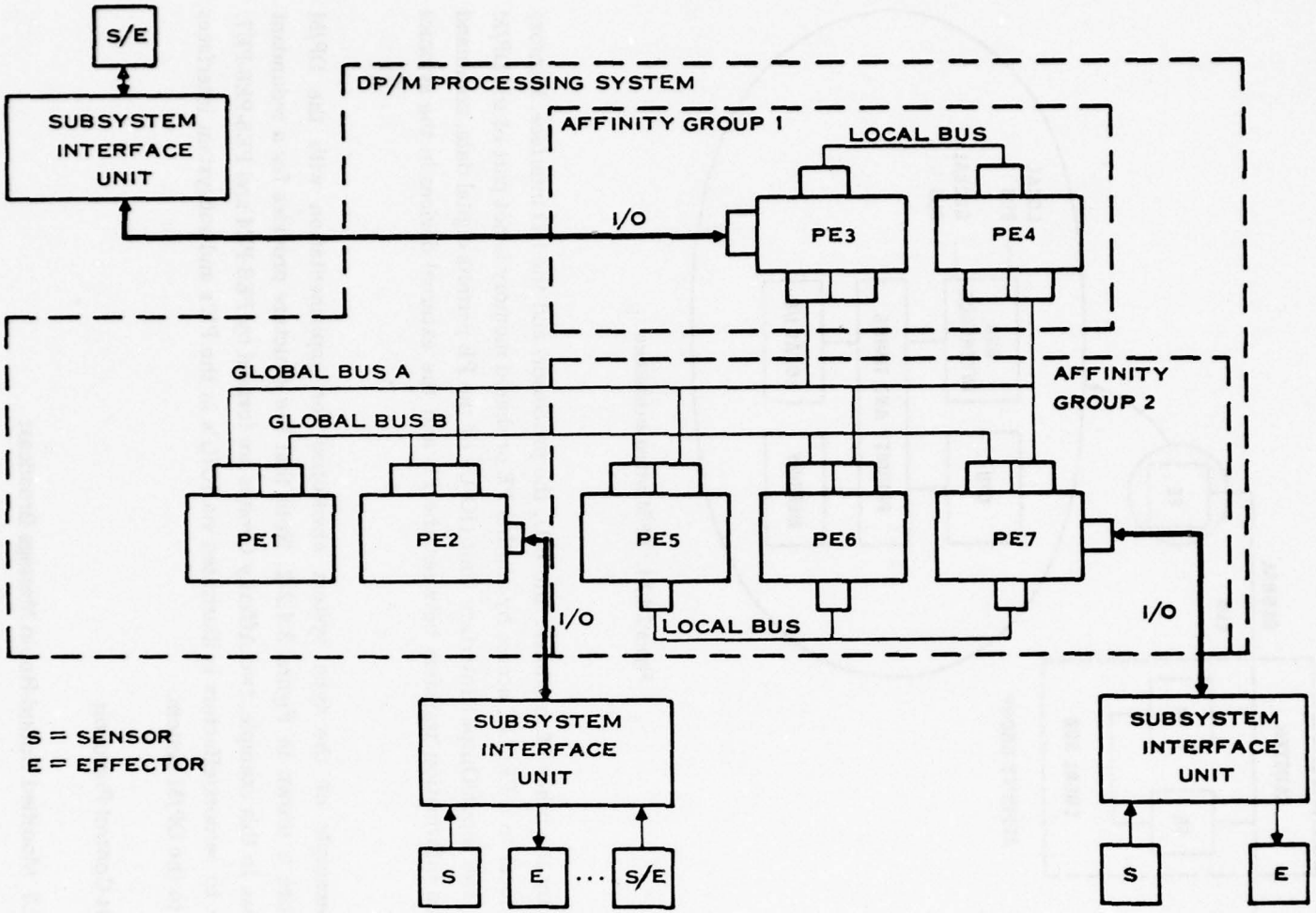


Figure 2.4.2-2. Example DP/M application.

provides for simple advancing of the "bus control slot" from PE to PE in a predetermined order among the total set of PE's attached to the bus. Each bus transmission, referred to as a message, is terminated by the transmitting PE. Message termination is denoted by placing an end-of-message, or message-sync signal on the bus immediately following the last data bit in the message. If a PE cannot transmit data when it receives bus access, it merely "passes" control to the next subsequent bus-controlling PE by transmitting on the message-sync signal. A variable length message may be transmitted, although the maximum length of a Global bus message is eight data words. Messages transmitted by any PE can be received by any other PE, since PE's are always in a "listening" mode when not transmitting (at all times other than its bus slot). This "broadcast" method is implemented, as described in subsequent sections, by providing unique input/output message designators for each PE.

2.4.3.2 MIL-STD-1553A

The second bus protocol algorithm implemented by DP/M SNS is that specified by MIL-STD-1553A (Aircraft Internal Time Division Command/Response Multiplex Data Bus). The basic difference between the 1553A protocol and the round-robin protocol is that data transfers occur only between two PE's, one specifically designated as a transmitter and the other as a receiver. Three word formats are defined for the protocol: (1) Command word, (2) Data word, and (3) Status word. One PE must be designated as a bus controller as shown in Figure 2.4.3-1. Note that the Remote Terminals (RT's) and Bus Controller each correspond to a PE in the architecture of Figure 2.4.2-1. Sole control of information transmission on the bus resides with the controller, which initiates all transmissions. Transmissions are performed in a half-duplex, asynchronous manner. Three message formats are permitted:

1. Controller to RT transfers,
2. RT to controller transfers, and
3. RT to RT transfers.

In controller to RT transfers, the controller issues a receive command to an RT followed by the specified number of data words. The RT, after message validation, transmits a status word back to the controller. For RT controller transfers, the controller issues a transmit command to the RT. After command verification, the RT transmits a status word back to the controller followed by the data words. In the case of RT to RT transfers, the controller issues a receive command to RT-A, followed by a transmit command to RT-B. The RT-B then transmits the same format as in controller to RT transfers.

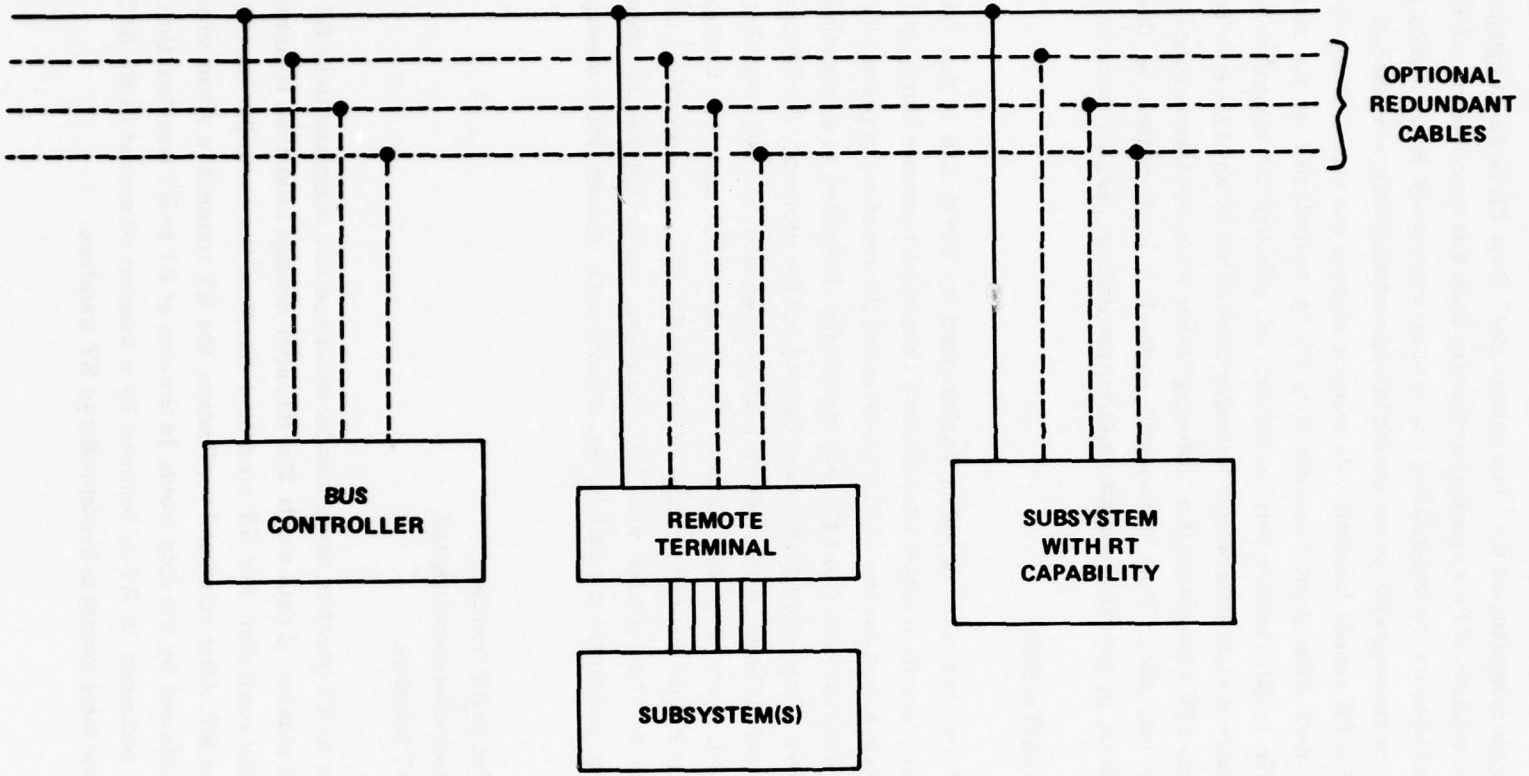


Figure 2.4.3-1. 1553A multiplex data bus architecture.

The implementation of both bus protocol schemes, round-robin and 1553A, provides the system designer with the opportunity to explore some basic tradeoffs in bus traffic overhead versus flexibility and assignment of executive functions to PE's, among others. Included in the tradeoffs are the inherent architectural differences of the two bus protocols. The 1553A, for example, does not allow local buses.

2.4.4 Avionics Software

The hardware architecture represents one-half of the distributed avionics system design problem, the other half is obviously the software. The DP/M SNS assumes that the system designer will partition the executive and applications software into suitable tasks for some given hardware architecture. The SNS then provides the designer with the capability to analyze and evaluate:

1. Processing element characteristics/capabilities,
2. Number of resources in the system,
3. Local and global bus configuration,
4. Inter-PE communication technique,
5. PE Bus protocol communication technique, and
6. Executive control technique.

2.4.4.1 Applications Software Functions

The SNS is predicated on the representation of software modules by a directed graph, such as Figure 2.4.4.1-1. A directed graph consists of a set of nodes and a set of directed edges between these nodes. A node is used to represent a set of computations which, once initiated, can run to completion without waiting for completion of another set of computations also represented by a node. An edge from node i to node j means that, upon completion of the computations represented by node i, the computations by node k can be initiated. The implementation of a conditional branch in this model is accomplished by an exclusive-OR symbol ⊕. Use of the symbol in conjunction with node i implies one of the following conditions:

Upon completion of the computations associated with node i, only one of the set of successor nodes can be initiated.

Completion of only one of the nodes associated with an incident edge is necessary for initiation of node i.

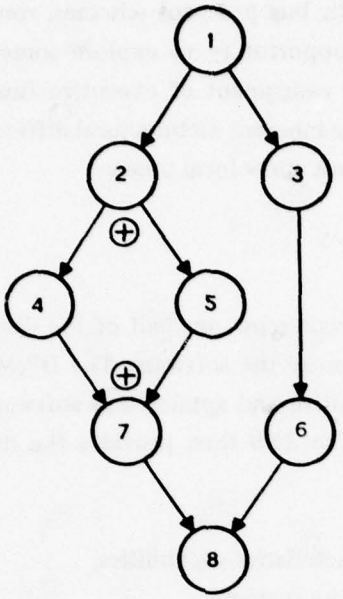


Figure 2.4.4.1-1. Example of directed graph.

In Figure 2.4.4.1-1, completion of node 1 indicates that nodes 2 and 3 can be initiated. Upon completion of node 2, either node 4 or node 5 can be initiated, and the completion of only one of these is sufficient to initiate node 7. Node 8, on the other hand, requires the completion of both node 6 and node 7 for its initiation. Stated differently, nodes 2 and 3 are successors of node 1, and nodes 6 and 7 are predecessors of node 8.

With respect to avionic mission processing, three levels of directed graphs can be used. At the highest level, a node represents a mission- or pilot-oriented function such as navigation. For any given function, a set of nodes or options may exist to effect the function. These nodes are referred to as subfunctions (e.g., navigation modes can include inertial navigation, Loran, or doppler navigation). The intent of the subfunction definition is to coincide with the normal division and classification of avionics computer programs. At the lowest level, a subfunction is represented by a set of related tasks. In the DP/M Executive structure, a task represents an executable application software module.

The representation of software execution sequences via a directed graph concept has a particular advantage within the DP/M concept. The subfunction directed graph reveals potential process construction options in allocating tasks and program to PE's. If any one set of tasks must be partitioned among several PE's, the options available in allocating this

software to PE's are clearly defined within the graph. Data sets that are passed from one task to another represent Local bus messages if their respective tasks are not collocated in the same PE. Likewise, collocated tasks need not generate bus traffic with their data interchanges.

The software task representation convention used by SNS is shown in Figure 2.4.4.1-2.

Each task is assigned a unique I.D. number, P-(N). The task execution time must be estimated from knowledge of the task size and processor speed or known from actual tests. The frequency of execution (cycle period) is also specified. All intertask information received by the task remove spare or generated by the task is defined in terms of a message number and the number of words in the message. In addition, local I/O is also specified by the number of words in the message.

The DP/M simulator assumes that the system designer will partition his system along the lines of mission functions and subfunctions, although this need not necessarily be the case. For example, such a set of functions might be:

1. Navigation,
2. Landing,
3. Air Data,
4. Flight Control,
5. Fire Control,
6. Vehicle Defense,
7. Stores Management, and
8. System Management.

These functions might be further divided into subfunctions, where Navigation might include:

1. Loran,
2. Inertial strapdown,
3. Kalman Filter,
4. Air Mass,
5. Wind Estimate, and
6. Steering.

The Loran subfunction might have tasks to simulate:

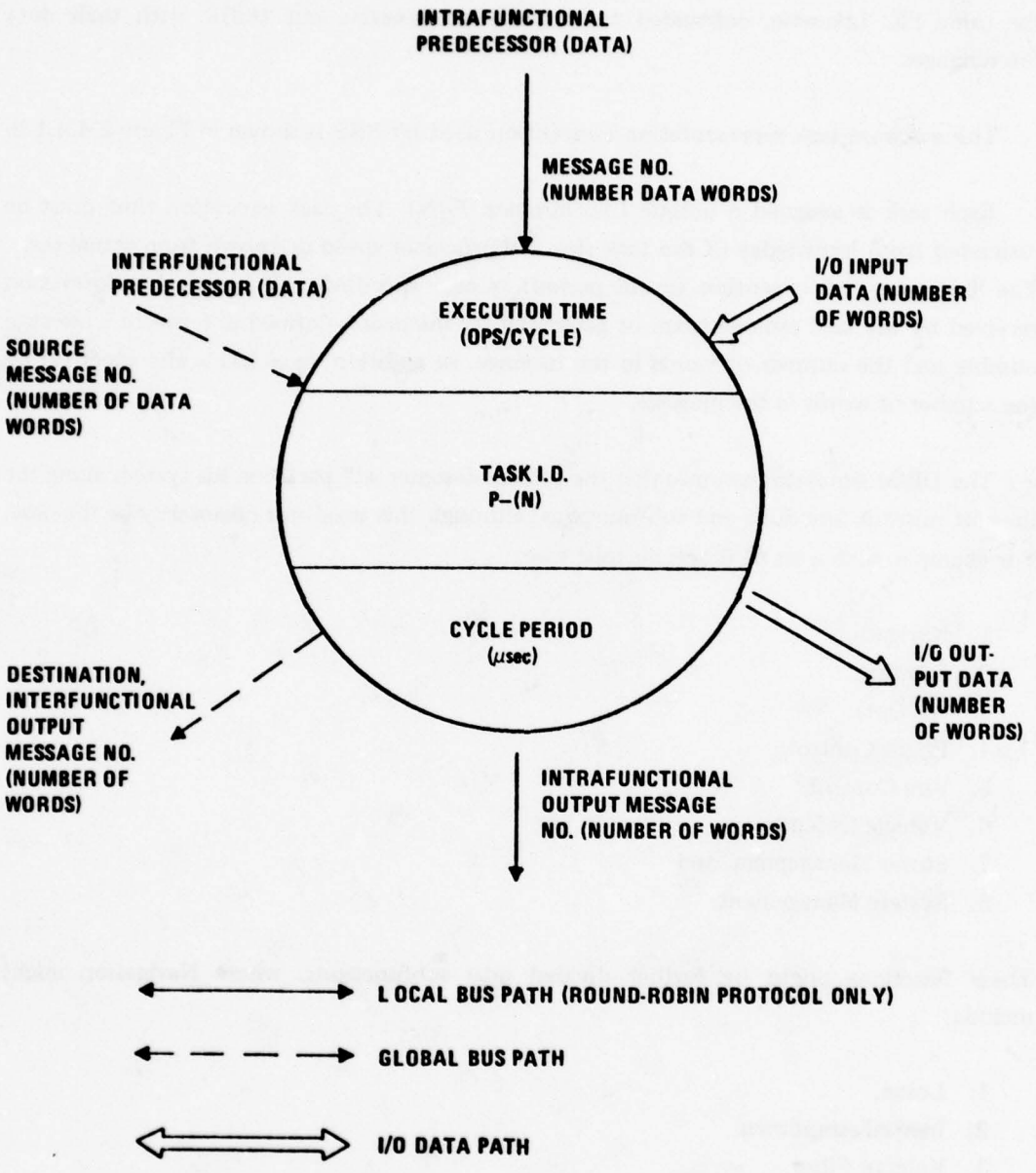


Figure 2.4.4.1-2. Directed graph task representation.

1. Receiver interface (P-11),
2. Bearing angle/expected time (P-12) difference,
3. Time difference to Lat./Long. (P-13),
4. Delay time correction/velocity information (P-14), and
5. Universal Transverse Mercator/Output Display (P-15).

The directed graph for the Loran task might be as shown in Figure 2.4.4.1-3.

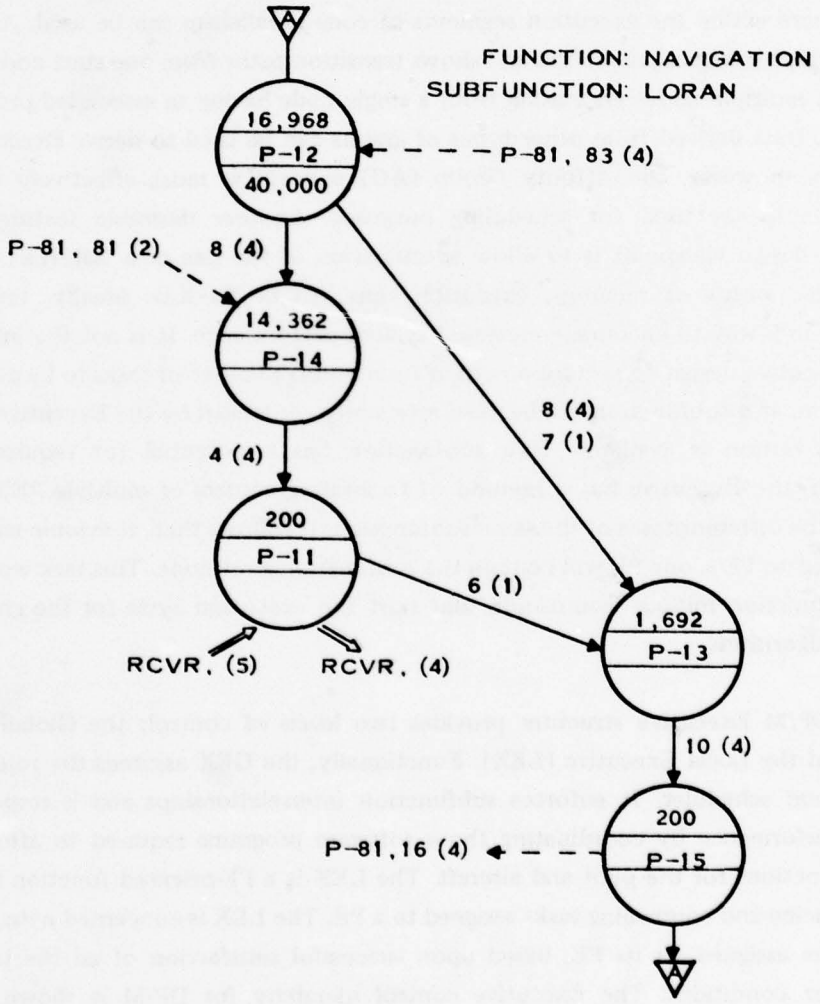


Figure 2.4.4.1-3. Loran subfunction directed graph.

2.4.4.2 Executive Software

The use of distributed Processing Elements (PE's) in the DP/M system concept dictates the need for a method of scheduling activities, transferring bus messages between PE's and general system control. These operations are referred to as Executive functions and are provided by the SNS.

The subfunction directed graph (Figure 2.4.4.1-3) contains the necessary information from which the Executive can determine task-scheduling conditions (based upon required predecessor events) and inter-task communication. Two types of directed graphs can be used to describe avionic functions: the sequential graphs and the parallel graph. A parallel graph reveals where within the execution segments of code parallelism can be used. A sequential program graph removes parellelism and shows transition paths from one start node to an end node with multiple nodes emanating from a single node having an associated probability of transition. Data derived from other types of graphs can be used to derive Executive design parameters; however, the Affinity Group (AG) concept is most effectively used when parallel graphs are used for scheduling purposes. Another desirable feature from the Executive design viewpoint is to allow specification of the size of a task (i.e., number of instructions, words of memory, execution time) to be flexible. Ideally, tasks will be identified in a way to encourage increased system performance. It is not the intent of the DP/M Executive design to require a certain (minimum) number of tasks to be created for a subfunction. If a subfunction can be most efficiently controlled by the Executive as a single task, this option is available. If a subfunction has a potential (or requirement) for parallelism, the Executive has a method of facilitating control of multiple PE's executing code for the different tasks of the same subfunction. It follows that, as avionic subfunctions are assigned to PE's, one PE will contain the subfunction start-node. This task would receive any "subfunction initiate' commands, and start the execution cycle for the given avionic software algorithms.

The DP/M Executive structure provides two levels of control: the Global Executive (GEX) and the Local Executive (LEX). Functionally, the GEX assumes the role of system monitor and scheduler. It enforces subfunction interrelationships and is responsible for system performance by coordinating those software programs required to affect mission avionic functions for the pilot and aircraft. The LEX is a PE-oriented function responsible for sequencing and controlling tasks assigned to a PE. The LEX is concerned with scheduling those tasks assigned to its PE, based upon successful satisfaction of all the tasks' given predecessor conditions. The Executive control hierarchy for DP/M is shown in Figure 2.4.4.2-1. This figure does not represent the individual routines within the Executive; rather, it shows the inter-relationships between major functions in the executive control hierarchy.

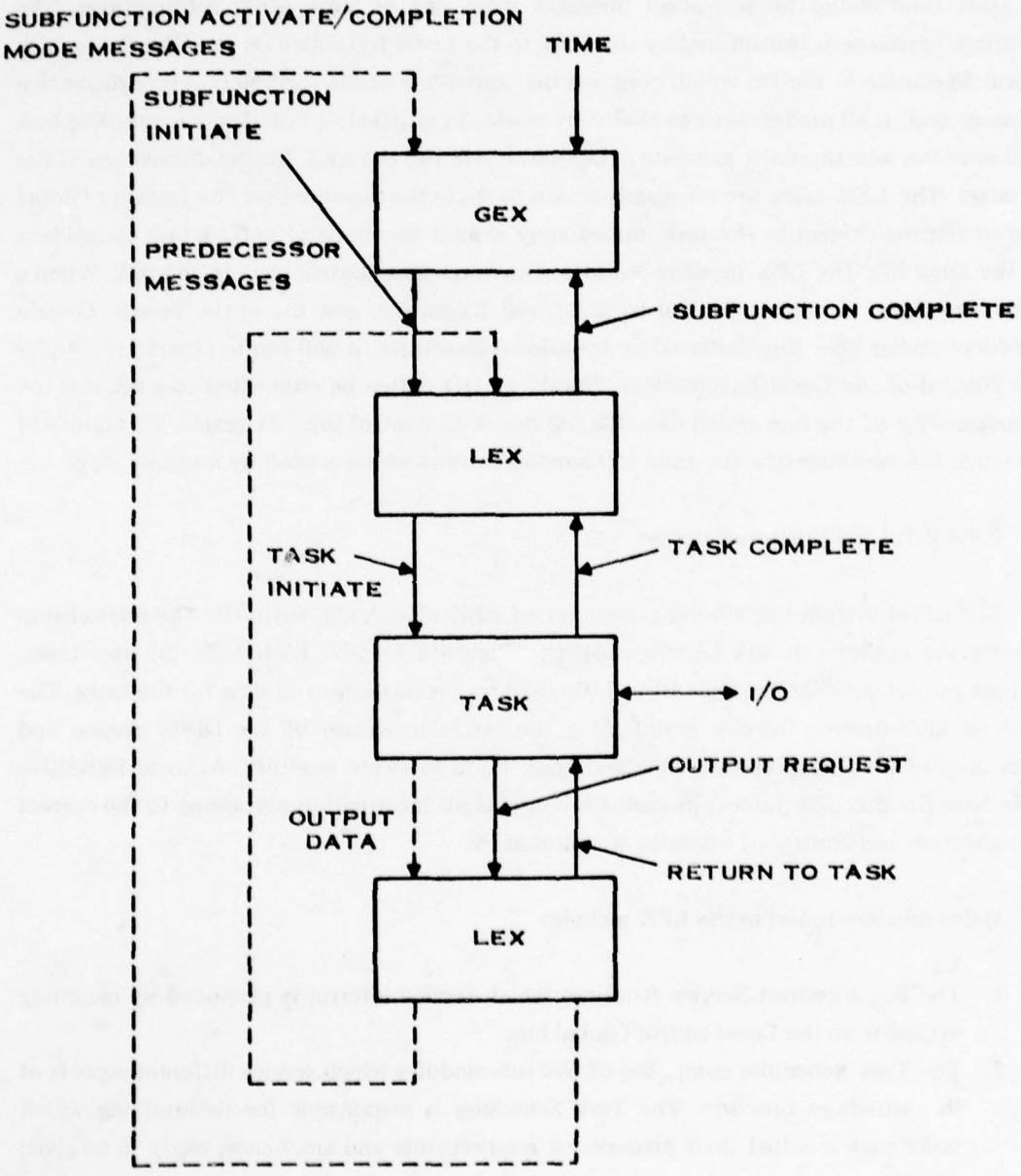


Figure 2.4.4.2-1. Executive control hierarchy.

The Global Executive initiates a subfunction when all the subfunction predecessor requirements are satisfied and when it is time for it to run. These predecessors, in addition to start time would be activation messages from one or more other subfunctions. The "initiate" message is transmitted by the GEX to the Local Executive on the Global bus. The Local Executive in the PE which contains the start node of the subfunction recognizes this message and, if all predecessors to that start mode are satisfied, it initiates the task. The task will execute, and should it generate a data set, it will call the LEX for the disposition of the data set. The LEX takes the necessary action to route the message over the Local or Global bus or returns control to the task immediately should the message be for a task co-resident in the same PE. The LEX message handler always releases control back to the task. When a task completes, it returns control to the Local Executive, and the cycle repeats. Once a subfunction has been time initiated by the Global Executive, it will run to completion under the control of the Local Executive(s). Should an I/O device be connected to a PE, it is the responsibility of the task which uses this I/O device to control this I/O device. This does not preclude the existence of a common I/O handler routine which is used by multiple tasks.

2.4.4.2.1 LEX functional design

The DP/M system has a homogenous set of LEX modules in each PE. The relationship among the modules of the LEX is shown in Figure 2.4.4.2-2. Each LEX initiates tasks, honors output message requests from tasks, and routes messages and data for the tasks. The LEX is table-driven, thereby maintaining the modular nature of the DP/M system and providing separation of system logic and application software modules. A Local Executive data base (or the LEX tables) in each PE contains all information pertaining to the correct initialization and control of the tasks within that PE.

Major routines found in the LEX include:

1. The Bus Interrupt Service Routines which service interrupts produced by incoming messages on the Local and/or Global bus.
2. The Task Scheduler composed of five sub-modules which service different aspects of the scheduler function. The Task Scheduler is responsible for determining which tasks have satisfied their predecessor requirements and are, hence, ready to be given control.
3. The Dispatcher module which transfers control to the highest priority task in the dispatcher queue.
4. The Output Message Interpretor module which is called by an applications task when it needs output message service. This module returns control to the task after servicing the request.

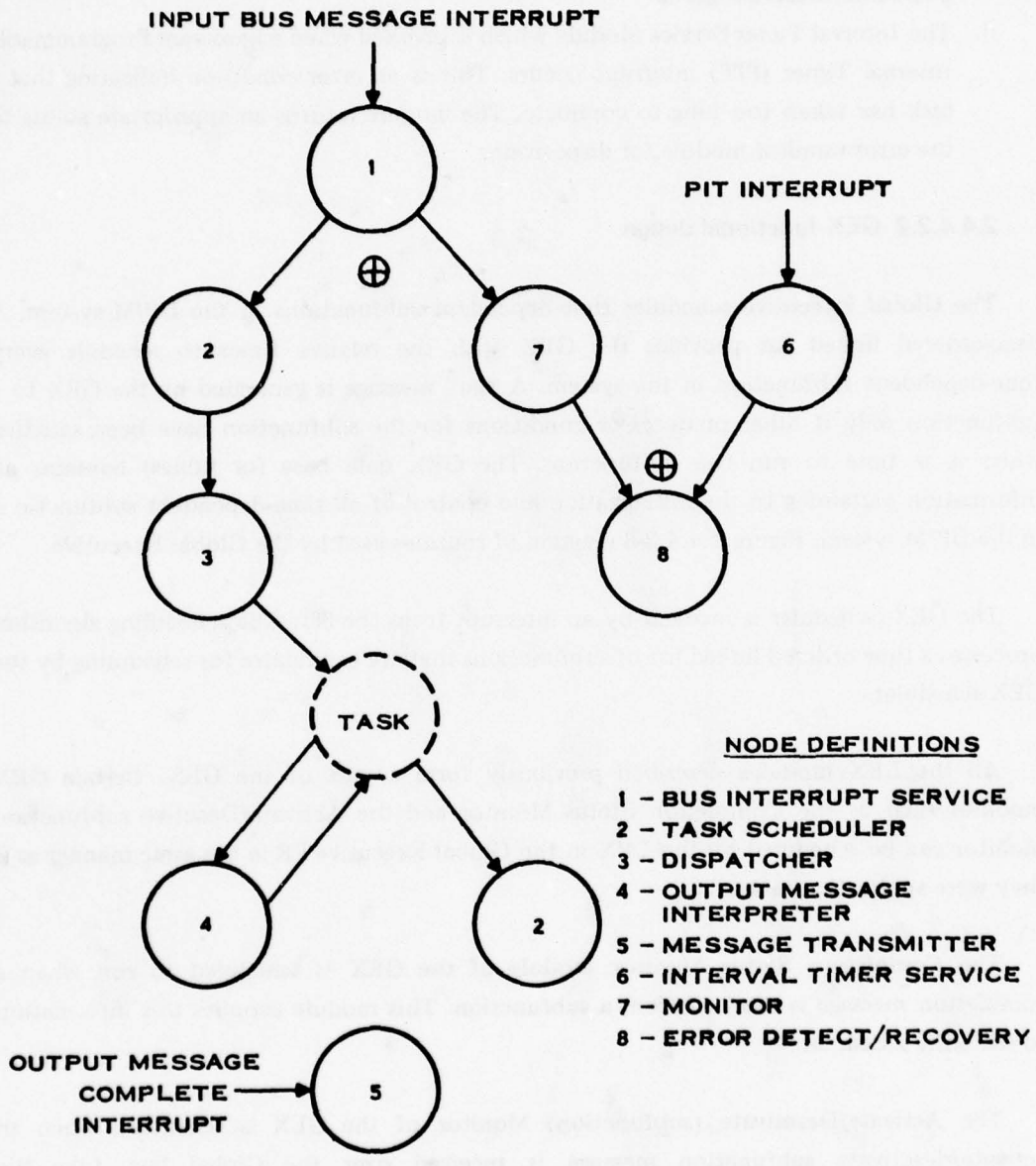


Figure 2.4.4.2-2. LEX module and task interrelationship.

5. The Message Transmitter Module which is invoked by the Output Message Complete Interrupt from one of the bus interfaces. The module is able to output data autonomously over the appropriate Local or Global bus should such a request be posted in its service queue.
6. The Interval Timer Service Module which is invoked when a hardware Programmable Internal Timer (PIT) interrupt occurs. This is an error condition indicating that a task has taken too long to complete. The module returns an appropriate status to the error monitor module for disposition.

2.4.4.2.2 GEX functional design

The Global Executive schedules time-dependent subfunctions in the DP/M system. A time-ordered linked list provides the GEX with the relative times to schedule every time-dependent subfunction in the system. A "go" message is generated by the GEX to a subfunction only if other predecessor conditions for the subfunction have been satisfied when it is time to run the subfunction. The GEX data base (or tables) contains all information pertaining to the initialization and control of all time-dependent subfunctions in the DP/M system. Figure 2.4.4.2-3 diagram of routines used by the Global Executive.

The GEX Scheduler is invoked by an interrupt from the PIT. The scheduling algorithm processes a time-ordered linked list of subfunctions that are candidates for scheduling by the GEX scheduler.

All the LEX modules described previously form a part of the GEX. Certain GEX modules such as the Completion Status Monitor and the Activate/Deactive subfunction monitor can be scheduled by the LEX in the Global Executive PE in the same manner as if they were application tasks.

The Completion Status Monitor module of the GEX is scheduled to run when a completion message is received from a subfunction. This module supplies this information to the GEX scheduler.

The Activate/Deactivate (subfunction) Monitor of the GEX is scheduled when an activate/deactivate subfunction message is received over the Global bus. Like the Completion Status Monitor, the activate/deactivate monitor supplies this information to the GEX scheduler.

The Mode Change Detector module is scheduled when a mode change command is

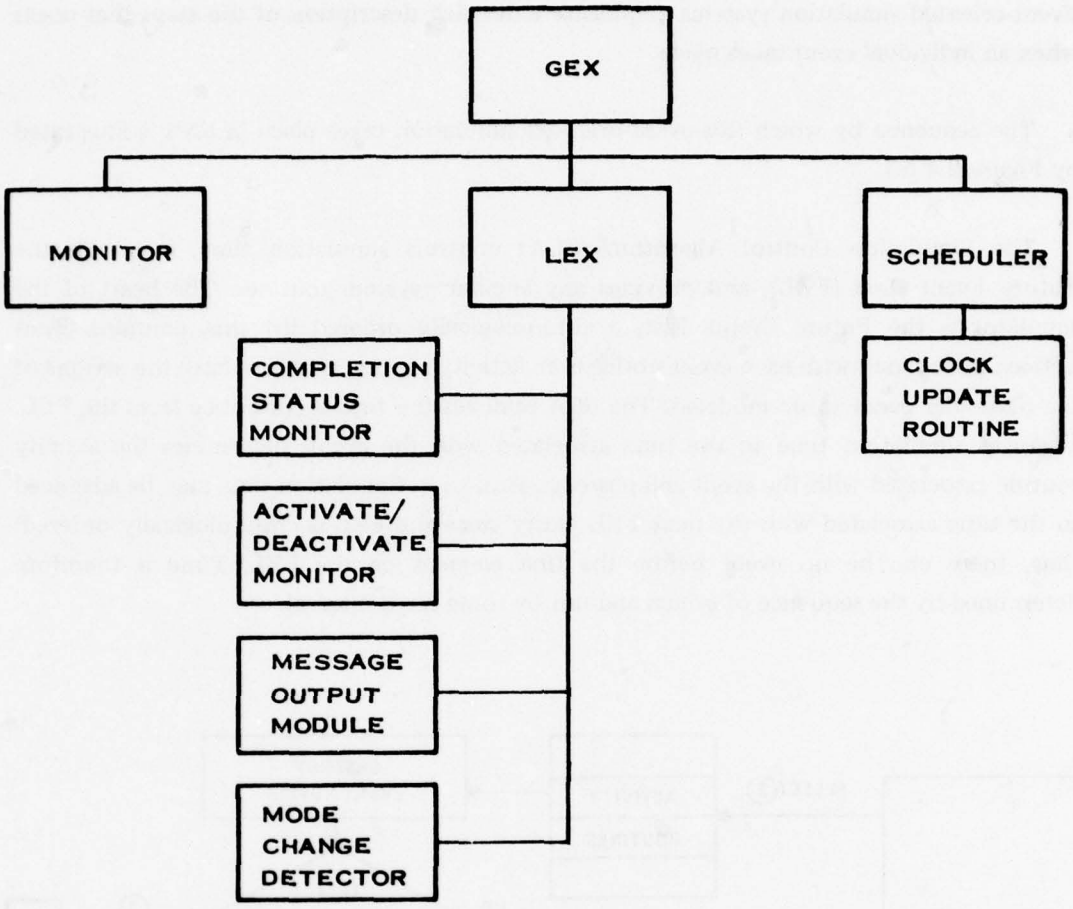


Figure 2.4.4.2-3. GEX block diagram.

received over the Global bus. A mode change command provides a quick method of initiating a new set of subfunctions such as might be involved when the pilot selects a master function switch on a control panel and causes multiple subfunctions to become active.

2.4.5 DP/M SNS Simulation Control

The SNS, being constructed of SIMNUC model independent routines, has the same general characteristics as SIMNUC and GASP IV. That is, SNS is a discrete event-oriented simulation system. Unlike a continuous system where transitions from one state to the next are a continuous function of time, transitions from one state to another in a discrete system occur at discrete points in time. Distinguishable state transitions are called events.

Event-oriented simulation systems emphasize a detailed description of the steps that occur when an individual event takes place.

The sequence by which this event-oriented simulation takes place in SNS is illustrated by Figure 2.4.5-1.

The Simulation Control Algorithm (SCA) controls simulation time, maintains the Future Events List (FEL), and provides any ancillary system routines. The heart of the simulator is the Future Events List, a chronologically ordered list that contains event notices. Associated with each event notice is an activity routine that simulates the actions of the particular event to be modeled. The SCA removes the first event notice from the FEL, advances simulation time to the time associated with the event, determines the activity routine associated with the event and passes control to it. Simulation time may be advanced to the time associated with the next FEL entity since the FEL is chronologically ordered; thus, there can be no event before the first element on the FEL. Time is therefore determined by the sequence of events and not by some fixed interval.

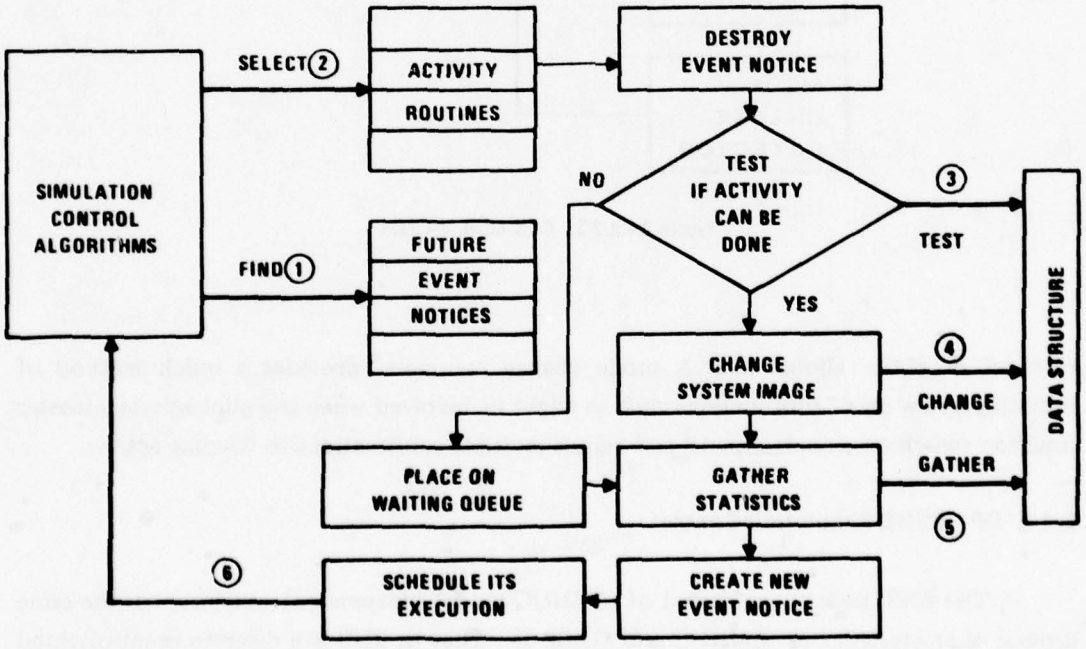


Figure 2.4.5-1. Simulation control structure.

Each activity routine performs essentially the same task. First, it destroys its event notice, after extracting any pertinent information, by returning it to dynamic memory. Next, the activity must determine, via its data structures, if all precedence relationships have been satisfied. If not, the activity is placed in a waiting queue until such time as the constraints are satisfied. If all constraints are satisfied, the activity may be executed, thus changing the state of the system. Necessary statistics are then gathered. Finally, a new event notice is generated, a time for the event to be executed in the future is computed, and the next event notice is returned to the SCA to be placed on the FEL. The SCA then removes the next event from the FEL and proceeds in the same manner as before.

2.4.6 DP/M SNS Reporting Capability

A family of data collection and report generation programs are provided with the DP/M System Network Simulator. These programs provide the capability to selectively collect data on and generate reports for the various system parameters under investigation for a particular DP/M system configuration and/or avionic mission segment. Both the collection and dispensation of data as well as the generation of reports are controlled by user specified parameters. In general, the user has four options: (1) no data is collected and no reports generated, (2) data is collected and saved, but no report generated, (3) data is collected and report generated but data not saved, (4) data is collected and saved, and reports generated. Data saved on magnetic tape may be processed at a later time. In fact, this saved data may be used at a later time to compare results of two or more different simulation experiments. The particular options available are discussed in detail in the following sections.

Data collection and report generation occur at three distinct levels: (1) event level, (2) sample period level, (3) post simulation level. At each of these levels, reports concerning bus performance, processor loading, executive performance, and number of avionic tasks processed may be selectively generated.

2.4.6.1 Event Level Reports

Event level reports consist of those reports that are generated at the completion of each simulation event. An event level report provides a single line output that indicates the current event execution. These reports are used to provide an event-by-event trace of the flow of the simulation throughout a particular simulation experiment. For example:

** EVENT ROUTINE "NAME" AT TIME "NOW" PROCESSING "N"

is an event level report. "NAME" is the name of the event routine currently executing. "NOW" is current simulation time. "N" is the identification of the entity currently being processed by the event. N is normally an avionic task or message identification number.

2.4.6.2 Sample Period Reports

Sample period reports consist of those reports that are generated at the completion of a user specified sample time. Data are collected during a period of simulation time (sample period) and then reports (if specified) are generated at the completion of that time. Figure 2.4.6.2-1 is an example of a sample period report for a local bus. Figure 2.4.6.2-2 is an example of a sample period report for a processor. The user may specify which reports are to be generated.

In all report examples, times are measured in seconds. A minor frame is the user defined sample period. A major frame is some user defined sample period that contains an integer number of minor frames. Message numbers are those identifications assigned by the system test data. Message lengths are measured in bits including all overhead bits. Origin and destination are the message origin and destination processor identification numbers. Relative time of start is measured from the start of the minor frame. Bus utilization is measured as a percentage of total available bus bandwidth.

2.4.6.3 Post Simulation Reports

Post simulation reports consist of those summary type reports that are generated after the completion of a particular simulation experiment. Post simulation reports provide a concise summary of all data collected throughout the simulation. These summaries allow rapid evaluation of simulation results without massive data reduction. If after initial evaluation it is necessary to investigate in further detail, sample period or even event level reports may be utilized. The user may specify explicitly which reports are to be generated even though data were collected on all entities. Figure 2.4.6.3-1 is an example of the bus decomposition report. The bus decomposition report shows the relative usage of the bus bandwidth by each component of a message. Figure 2.4.6.3-2 is an example of the bus loading bar graph. This graph shows the percentage of bus utilization during each sample period. Figure 2.4.6.3-3 is an example of the bus utilization summary report. This report summarizes the performance of each bus in the DP/M system during the complete simulation experiment. This report is actually a summary of the sample period bus activity reports. Also associated with the bus usage summary is a summary of all messages transferred over that particular bus. Figure 2.4.6.3-4 is an example of the message utilization

```
**********************************************************************
OPM SIMULATION TEST CASE                                  12:18:40   05/28/74
BUS ACTIVITY RECORD FOR BUS NUMBER          1
MAJOR FRAME NUMBER    1                              MINOR FRAME NUMBER    1
----------------------------------------------------------------------

MINOR FRAME START TIME           =    0.0
TOTAL BUS USAGE TIME             =    0.000276
PERCENT OF MINOR FRAME UTILIZED  =    5.52

----------------------------------------------------------------------

NUMBER OF MESSAGES TRANSMITTED   =    5
MAXIMUM TIME ON OUTPUT QUEUE     =    0.000164          FOR MESSAGE        400
MINIMUM TIME ON OUTPUT QUEUE     =    0.000012          FOR MESSAGE         22
AVERAGE TIME ON OUTPUT QUEUE     =    0.000080
LENGTH OF LONGEST MESSAGE TRANSMITTED  =         70
LENGTH OF SHORTEST MESSAGE TRANSMITTED =         37
AVERAGE LENGTH OF MESSAGE TRANSMITTED  =      50.00
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER       22
MESSAGE LENGTH =        70     ORIGIN =       1     DESTINATION =          2
RELATIVE TIME OF START =     0.000312    LENGTH OF TRANSMISSION =     0.000075
TIME ON OUTPUT QUEUE   =     0.000012
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER      400
MESSAGE LENGTH =        37     ORIGIN =       1     DESTINATION =          2
RELATIVE TIME OF START =     0.000403    LENGTH OF TRANSMISSION =     0.000042
TIME ON OUTPUT QUEUE   =     0.000103
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER       24
MESSAGE LENGTH =        70     ORIGIN =       2     DESTINATION =          3
RELATIVE TIME OF START =     0.002735    LENGTH OF TRANSMISSION =     0.000075
TIME ON OUTPUT QUEUE   =     0.000015
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER      400
MESSAGE LENGTH =        37     ORIGIN =       2     DESTINATION =          1
RELATIVE TIME OF START =     0.002826    LENGTH OF TRANSMISSION =     0.000042
TIME ON OUTPUT QUEUE   =     0.000106
**********************************************************************
MESSAGE DESCRIPTOR FOR MESSAGE NUMBER      400
MESSAGE LENGTH =        37     ORIGIN =       2     DESTINATION =          3
RELATIVE TIME OF START =     0.002884    LENGTH OF TRANSMISSION =     0.000042
TIME ON OUTPUT QUEUE   =     0.000164
**********************************************************************
```

Figure 2.4.6.2-1. Sample period bus report.

```
**************************************************************************
DPM SIMULATION TEST CASE                                  12:18:40   05/28/74
COMPUTER UTILIZATION FOR PROCESSOR 2
--------------------------------------------------------------------------
MAJOR FRAME NUMBER =     1                  MINOR FRAME NUMBER =     1
REPORT START TIME  =     0.0                REPORT STOP TIME   =     0.005000
--------------------------------------------------------------------------
                          TIME UTILIZATION DATA
NUMBER OF INTERRUPTS SERVICED   =          0
TOTAL INTERRUPT SERVICE TIME    =     0.0                  PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.000300             PERCENT =    6.00
TOTAL APPLICATIONS PROGRAM TIME =     0.002270             PERCENT =   45.40
TOTAL IDLE TIME                 =     0.002430             PERCENT =   48.60
--------------------------------------------------------------------------
                          MEMORY UTILIZATION DATA
AVAILABLE MEMORY =     4000
MAXIMUM USED     =        0                        AVERAGE USED =         0
--------------------------------------------------------------------------
                          MESSAGE UTILIZATION DATA
NUMBER OF INCOMING MESSAGES =          3     INCOMING MESSAGES MISSED =     0
NUMBER OF OUTGOING MESSAGES =          3     OUTGOING MESSAGES MISSED =     0
**************************************************************************
```

Figure 2.4.6.2-2. Sample period processor report.

```
**********************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS    1
----------------------------------------------------------------------
TOTAL TRAFFIC                  =      20.96 KBPS
TRAFFIC UTILIZED FOR DATA      =      15.60 KBPS   74.41 PERCENT
TRAFFIC UTILIZED FOR HEADER    =       2.26 KBPS   10.77 PERCENT
TRAFFIC UTILIZED FOR SYNC      =       1.16 KBPS    5.56 PERCENT
TRAFFIC UTILIZED FOR GAP       =       1.94 KBPS    9.26 PERCENT
**********************************************************************
**********************************************************************
                BUS TRAFFIC DECOMPOSITION FOR LOCAL  BUS    2
----------------------------------------------------------------------
TOTAL TRAFFIC                  =       0.0  KBPS
**********************************************************************
**********************************************************************
                BUS TRAFFIC DECOMPOSITION FOR GLOBAL BUS   99
----------------------------------------------------------------------
TOTAL TRAFFIC                  =      25.24 KBPS
TRAFFIC UTILIZED FOR DATA      =      22.60 KBPS   89.56 PERCENT
TRAFFIC UTILIZED FOR HEADER    =       1.51 KBPS    5.97 PERCENT
TRAFFIC UTILIZED FOR SYNC      =       0.42 KBPS    1.68 PERCENT
TRAFFIC UTILIZED FOR GAP       =       0.71 KBPS    2.80 PERCENT
**********************************************************************
```

Figure 2.4.6.3-1. Bus decomposition report.

```
DPM SIMULATION TEST CASE                                        10:45:35    03/03/74
.  FRAME        BUS ACTIVITY SUMMARY REPORT FOR BUS   1                    PAGE     1
. MAJ MIN       TIME    PERCENT UTILIZATION GRAPH                                    .
-----------------------0--------20---------40---------60---------80--------100
.   1   1     0.000000   5.52.888          .         .         .         .         .
.   1   2     0.005000   0.00.8            .         .         .         .         .
.   1   3     0.010000   3.18.88           .         .         .         .         .
.   1   4     0.015000   0.00.8            .         .         .         .         .
.   1   5     0.020000   3.18.88           .         .         .         .         .
.   1   6     0.025000   0.00.8            .         .         .         .         .
.   1   7     0.030000   3.18.88           .         .         .         .         .
.   1   8     0.035000   0.00.8            .         .         .         .         .
.   1   9     0.040000   5.52.888          .         .         .         .         .
.   1  10     0.045000   0.00.8            .         .         .         .         .
-----------------------0--------20---------40---------60---------80--------100
```

Figure 2.4.6.3-2. Bus loading bar graph.

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/76
                    BUS UTILIZATION SUMMARY FOR BUS    1                PAGE    1

REPORT START TIME =     0.0                          REPORT STOP TIME =     0.085000

TOTAL BUS USAGE TIME      =     0.001782
AVERAGE BUS UTILIZATION   =     2.10
MAXIMUM BUS UTILIZATION   =     5.52      DURING MAJOR FRAME    1 MINOR FRAME    1
MINIMUM BUS UTILIZATION   =     0.0       DURING MAJOR FRAME    1 MINOR FRAME    2

TOTAL NUMBER OF MESSAGES TRANSMITTED  =          33
MAXIMUM TIME ON OUTPUT QUEUE          =     0.000164      FOR MESSAGE          400
                                       DURING MAJOR FRAME    1 MINOR FRAME    1
MINIMUM TIME ON OUTPUT QUEUE          =     0.0           FOR MESSAGE           24
                                       DURING MAJOR FRAME    2 MINOR FRAME    7
AVERAGE TIME ON OUTPUT QUEUE          =     0.000055
LENGTH OF LONGEST MESSAGE TRANSMITTED =          70
                                       DURING MAJOR FRAME    1 MINOR FRAME    1
LENGTH OF SHORTEST MESSAGE TRANSMITTED =         37
                                       DURING MAJOR FRAME    1 MINOR FRAME    1
AVERAGE LENGTH OF MESSAGE TRANSMITTED =        49.00
```

Figure 2.4.6.3-3. Bus utilization summary report.

```
OPM SIMULATION TEST CASE                                    12:18:40    05/28/74
.                MESSAGE TRANSMISSION SUMMARY FOR BUS    1          PAGE     2

REPORT START TIME =     0.0                        REPORT STOP TIME =     0.005000

MESSAGE SUMMARY FOR MESSAGE NUMBER        22                                    .
NUMBER OF TIMES MESSAGE TRANSMITTED =      9                                    .
MESSAGE LENGTH =     70       LENGTH OF TRANSMISSION =     0.000075           .
ORIGIN          =     1                                                        .
DESTINATION     =     2                                                        .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000016                                     .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    3   .
MINIMUM TIME ON OUTPUT QUEUE =    0.000006                                     .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    3   .
AVERAGE TIME ON OUTPUT QUEUE =    0.000010                                     .
*****************************************************************************
MESSAGE SUMMARY FOR MESSAGE NUMBER        24                                    .
NUMBER OF TIMES MESSAGE TRANSMITTED =      3                                    .
MESSAGE LENGTH =     70       LENGTH OF TRANSMISSION =     0.000075           .
ORIGIN          =     2                                                        .
DESTINATION     =     3                                                        .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000015                                     .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    1   .
MINIMUM TIME ON OUTPUT QUEUE =    0.0                                          .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    7   .
AVERAGE TIME ON OUTPUT QUEUE =    0.000006                                     .
*****************************************************************************
MESSAGE SUMMARY FOR MESSAGE NUMBER       400                                    .
NUMBER OF TIMES MESSAGE TRANSMITTED =     21                                    .
MESSAGE LENGTH =     37       LENGTH OF TRANSMISSION =     0.000042           .
ORIGIN          =     1    2                                                   .
DESTINATION     =     2    1    3                                              .
MAXIMUM TIME ON OUTPUT QUEUE =    0.000164                                     .
.                                 DURING MAJOR FRAME    1 MINOR FRAME    1   .
MINIMUM TIME ON OUTPUT QUEUE =    0.000003                                     .
.                                 DURING MAJOR FRAME    2 MINOR FRAME    1   .
AVERAGE TIME ON OUTPUT QUEUE =    0.000082                                     .
*****************************************************************************
```

Figure 2.4.6.3-4. Message transmission summary report.

summary report. Reports for each bus may be selected by the user. Figure 2.4.6.3-5 is an example of the processor utilization bar graph. This graph shows the percentage of processor utilization by interrupt servicing, system (executive) programs, and example of the processor utilization summary report applications programs. Figure 2.4.6.3-6 is an example of the processor utilization summary report. This report summarizes the utilization of each processor in the DP/M system during the simulation experiment. This report is actually a summary of the sample period processor usage reports.

2.4.7 Simulation Control Namelist Specifications

The user defines the architecture of a given system to the DP/M simulation through a set of namelist specifications. Simulation output information desired by the user in the form of reports (as described by Section 2.4.6) is specified through simulation control card images. These specifications are supplied to DP/M in the following order:

1. Report Control Specification Data,
2. Avionic Task Definition Data,
3. Bus Performance and Connectivity Definition Data,
4. Task to Processing Element Assignment, and
5. Subfunction Scheduling Definition Data.

Each of these input specifications is briefly described below for the case of the round robin bus protocol as an example of the user input required to run the DP/M SNS.

2.4.7.1 Report Control Specification Data

The card images for run control and report printing are given in Table 2.4.7.1-1. Data items are self-explanatory except for the following notes:

Card #2 Field #5 — This flag produces a trace print and memory dump of internal data at initialization.

Card #3 Field #6 — These numbers control a trace print each time an event routine of the corresponding PE is entered. Allows control of PE number 1 thru 16.

Card #3 Field #8 — These numbers indicate the writing mode selector for the various output records

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/74
.  FRAME        COMPUTER UTILIZATION REPORT FOR PROCESSOR 2          PAGE     1
. MAJ MIN        TIME    PERCENT UTILIZATION GRAPH                            .
                          0--------20--------40--------60--------80--------100
.   1   1      0.000000   51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   2      0.005000    0.00.          .         .       .         .         .
.   1   3      0.010000   51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   4      0.015000    0.00.          .         .       .         .         .
.   1   5      0.020000   51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   6      0.025000    0.00.          .         .       .         .         .
.   1   7      0.030000   51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1   8      0.035000    0.00.          .         .       .         .         .
.   1   9      0.040000   51.40.SSSAAAAAAAAAAAAAAAAAAAAA    .         .         .
.   1  10      0.045000    0.00.          .         .       .         .         .
                          0--------20--------40--------60--------80--------100
.   I = INTERRUPT SERVICE    S = SYSTEMS PROGRAM    A = APPLICATIONS PROGRAM    .
```

Figure 2.4.6.3-5. Processor utilization bar graph.

```
DPM SIMULATION TEST CASE                                    12:18:40    05/28/74
.              COMPUTER UTILIZATION SUMMARY FOR PROCESSOR 2              PAGE     1
-------------------------------------------------------------------------------
REPORT START TIME =     0.0                      REPORT STOP TIME =     0.085000
-------------------------------------------------------------------------------
.                          TIME UTILIZATION DATA                               .
NUMBER OF INTERRUPTS SERVICED   =            0                                 .
TOTAL INTERRUPT SERVICE TIME    =     0.0                    PERCENT =    0.0
TOTAL SYSTEMS PROGRAM TIME      =     0.002700               PERCENT =    3.18
TOTAL APPLICATIONS PROGRAM TIME =     0.020430               PERCENT =   24.04
TOTAL IDLE TIME                 =     0.061870               PERCENT =   72.79
-------------------------------------------------------------------------------
.                          MEMORY UTILIZATION DATA                             .
AVAILABLE MEMORY =      4000                                                   .
MAXIMUM USED     =         0                          AVERAGE USED =          0
-------------------------------------------------------------------------------
.                          MESSAGE UTILIZATION DATA                            .
NUMBER OF INCOMING MESSAGES =         29     INCOMING MESSAGES MISSED =        0
NUMBER OF OUTGOING MESSAGES =         21     OUTGOING MESSAGES MISSED =        0
*******************************************************************************
```

Figure 2.4.6.3-6. Processor utilization summary report.

TABLE 2.4.7.1-1. REPORT CONTROL SPECIFICATION DATA

Field No.	Columns	R/L Adj.	Punch Format	Description/Value
CARD #1				
1	1-8	L	A8	CARD NAME = 'CONTROL'
2	9-10	R	12	CARD SEQUENCE # = 01
3	11-12		2X	
4	13-80	L	17A4	RUN ID/DESCRIPTION
CARD #2				
1	1-8	L	A8	CARD NAME = 'CONTROL'
2	9-10	R	12	CARD SEQUENCE # = 02
3	11-12		2X	
4	13-24	L	3A4	ADDITIONAL RUN ID
5	25-30	R	15	DEBUG CONTROL FLAG (=1, TRUE)
6	31-35	R	15	
7	36-40		5X	
8	41-50	R	I10	SIMULATION START TIME
9	51-60	R	I10	SIMULATION END TIME
CARD #3				
1	1-8	L	A8	CARD NAME = 'CONTROL'
2	9-10	R	12	CARD SEQUENCE # = 03
3	11-20		10X	
4	21-36	R	1611	
5	37-40		4X	
6	41-56	R	1611	STCFLG FLAGS
7	57-60		4X	
8	61-76		1611	WRC FLAGS

0 = No print, no write to output file
1 = Print
2 = Write to output file
3 = Both

The columns are assigned to output reports as follows:

61 — Sample Period Bus Report
62 — Sample Period Processor Report
63 — Event Level Report
64 — Bus Decomposition Report
65 — Bus Utilization Summary Report
66 — Message Transmission Summary Report
67 — Processor Utilization Summary Report
68
.
. — Not Used
.
76

Observe that cards 1 and 2 identify the run and control the dump of initialization data. Card 3 controls the data collection and report printing.

2.4.7.2 Avionic Task Definition Data

Software tasks and their relationship to one another in terms of the directed graph and message structure described in Section 2.4.4 are specified by namelist data sets. All symbols start in column 2 of the data card image. Variable list values must be separated by commas with no imbedded blanks. Order within the namelist data sets is not important. Those variables with default values are optional and may be omitted. The namelist data items are given in Table 2.4.7.2-1.

Additional Executive Modules can be added to the SNS by defining them as tasks. In order to do so, executive modules are defined by task definition data sets as described by Table 2.4.7.2-1, task to PE assignment data sets as defined by Table 2.4.7.4-1, and, if necessary, subfunction scheduling data sets as shown in Table 2.4.7.5.1.

TABLE 2.4.7.2-1. AVIONIC TASK DEFINTION DATA

Variable	Description	Default Value/Restrictions
As many of the following data sets as there are tasks:		
$TDEFN	Start namelist data set descriptor	
LAST	=.True. If last task definition	
TASKID	Task identification number	
NPRED	Number of predecessors to task	0_{32}
DELTA	Inverse of rep. rate (period)	2^{-1}
TNAME=40H	Description of task (up to 40 characters)	*Blanks*
NUMSUC	Number of successor tasks	0
SRID	List of successor task IDs	
RTYPE	16-bit words of memory used	
NUMBS	Number of run time branch successors	0
BSID	List of run time branch successors	
NIMSG	Number of input message required by task	0
IMTYPE	List of input message types	
XTIME	Tasks execution time in processor clocks	0
$END	End namelist data set	
$MTBD	Start namelist data set	
NUMMSG	Number of output messages generated by task	
$END	End namelist data set	
As many of the following namelist data sets as specified by NUMMSG:		
$MDEFN	Start namelist data set	
TYPE	Message type	0
LENGTH	Message length in words exclusive of overhead	0
NUMDES	Number of destination tasks	0
TASKID	List of destination tasks	
PHASE	Transmission time	0
PERIOD	Period of message transmission	1
PRTY	Priority of message	0
CCLEN	Control length	17
SOURCE	Source ID	0
DENS	Density	0
$END	End namelist data set	

2.4.7.3 Bus Performance and Connectivity Definition Data

Specification of the number of processing elements (PE's) in the system, their identification and the bus performance is defined by user as in Table 2.4.7.3-1. Affinity groups are specified similarly as shown.

2.4.7.4 Task to Processing Element Assignment

Tasks to be executed by each processing element are assigned task by task as depicted in Table 2.4.7.4-1. The order of task execution in a given processor is resolved by the processor executive based on predecessor/successor relationships defined in Section 2.4.7.2.

TABLE 2.4.7.3-1. BUS PERFORMANCE AND CONNECTION DEFINITION DATA

Variable			Default Value
$GBDEF			
TOTPE	=	Total number of PE's in DP/M System	
GBCL	=	List of PE numbers that define the global bus connectivity in the order for control to be passed.	
BLGTH	=	Total number of elements in GBCL	
WSIZE	=	Bus data word length (equal to the sum of processor data word length plus error checking code length plus data word sync length)	16
BITPRD	=	Bus bit period (sec/bit)	1
MSYNC	=	Message sync signal length (bits)	3
GAPTME	=	Inter-message gap time (sec)	5
$END			
One set for each affinity group to be defined:			
$LBDEF			
NUMPE	=	Number of PE's in this affinity group	
PECON	=	List of PE numbers that define the local bus connectivity in the order for control to be passed	
BLGTH	=	Total number of elements in PECON	
LAST	=	.TRUE., use if last affinity group definition	
$END			

TABLE 2.4.7.4-1. TASK TO PROCESSING ELEMENT ASSIGNMENT DATA

One set for each processor assignment to be defined:

$DEFINE

PEID	=	Processor ID having tasks assigned to it	TASKS	=	List of tasks IDs to be assigned to the processor
NUMTSK	=	Number of tasks to be assigned to the processor	LAST	=	.TRUE., use if this is last processor assignment definition
			$END		

TABLE 2.4.7.5-1. SUBFUNCTION SCHEDULING DEFINITION DATA

As many of the following data sets as there are subfunctions:

$FNDEFN	Start namelist data set descriptor	SFPE	ID of PE's with subfunction start mode
RUNT	Time at which subfunction gets first 'go' message (μs)	LAST	=.TRUE. If last subfunction definition
ITER	Iteration period of subfunction (μs)	ID	Numeric identity of subfunction
NUMPE	Number of PE's with subfunction starting mode	$END	

2.4.7.5 Subfunction Scheduling Definition Data

Subfunction scheduling definition allows time-dependent subfunctions which are to be scheduled by the global executive to be defined as entries in a time-ordered list. Namelist specifications are given in Table 2.4.7.5-1.

SECTION 2.4 BIBLIOGRAPHY

Consolver, G., et al., *Distributed Processor/Memory Architecture Design Program*, Texas Instruments, Inc., Feb. 1975, AFAL-TR-75-80.

Texas Instruments, Inc., *User's Manual, Computer Program, DP/M System Network Simulation System*, Feb. 1975, Code Identification No. 96214, Unpublished.

2.5 SOFTWARE DESIGN AND VERIFICATION SYSTEM (SDVS)

2.5.1 SDVS Objectives

The Software Design and Verification System provides necessary non-real-time software tools to aid in the development, testing, verification, and maintenance of avionic mission software. The SDVS was created as an integral part of the Digital Avionic Information System (DAIS) program, and an understanding of the utility of SDVS is aided by an understanding of the objectives of DAIS. The DAIS concept proposes that the processing, multiplex, and display functions of avionic systems be common and service on an integrated basis all of the usual subsystem functions such as navigation, weapon delivery, stores management, flight controls, and communications. The DAIS program has attempted to demonstrate the feasibility of this concept in order to eliminate some of the proliferation of non-standard avionics and permit the Air Force to assume the initiative in the specification of future Air Force avionics.

Since the use of common hardware and software for acquisition of sensor data, processing of information, and provision of display information is critical to the DAIS concept (or to any other computer-oriented system concept), the software design and verification functions in the context of any particular processor architecture, are vital to success. The SDVS thus assumes a vital role in the design of future avionics systems. The following description attempts to describe SDVS in a general context rather than entirely within the DAIS framework, since it has a broad applicability to avionics system design.

2.5.2 SDVS Overview

The design of the SDVS is based on the program hierarchy shown in Figure 2.5.2-1. Note that the DEC system-10 operating system and utility routines are utilized by the SDVS. All of the SDVS programs are arranged in three levels. A program can be called only by a program of a higher level, and it, in turn, can call only programs at a lower level. Programs on the same level have no direct interface and must pass information through a program at the next higher level. The SDVS is written primarily in the JOVIAL-73 language and employs structured programming techniques. Other source languages are Cobol and Fortran.

The SDVS Control and Software Management Programs, though shown at the top of the hierarchy, are actually a collection of utility routines that are used by all the second level programs. The exception is the SDVS Control Program function that interacts with the

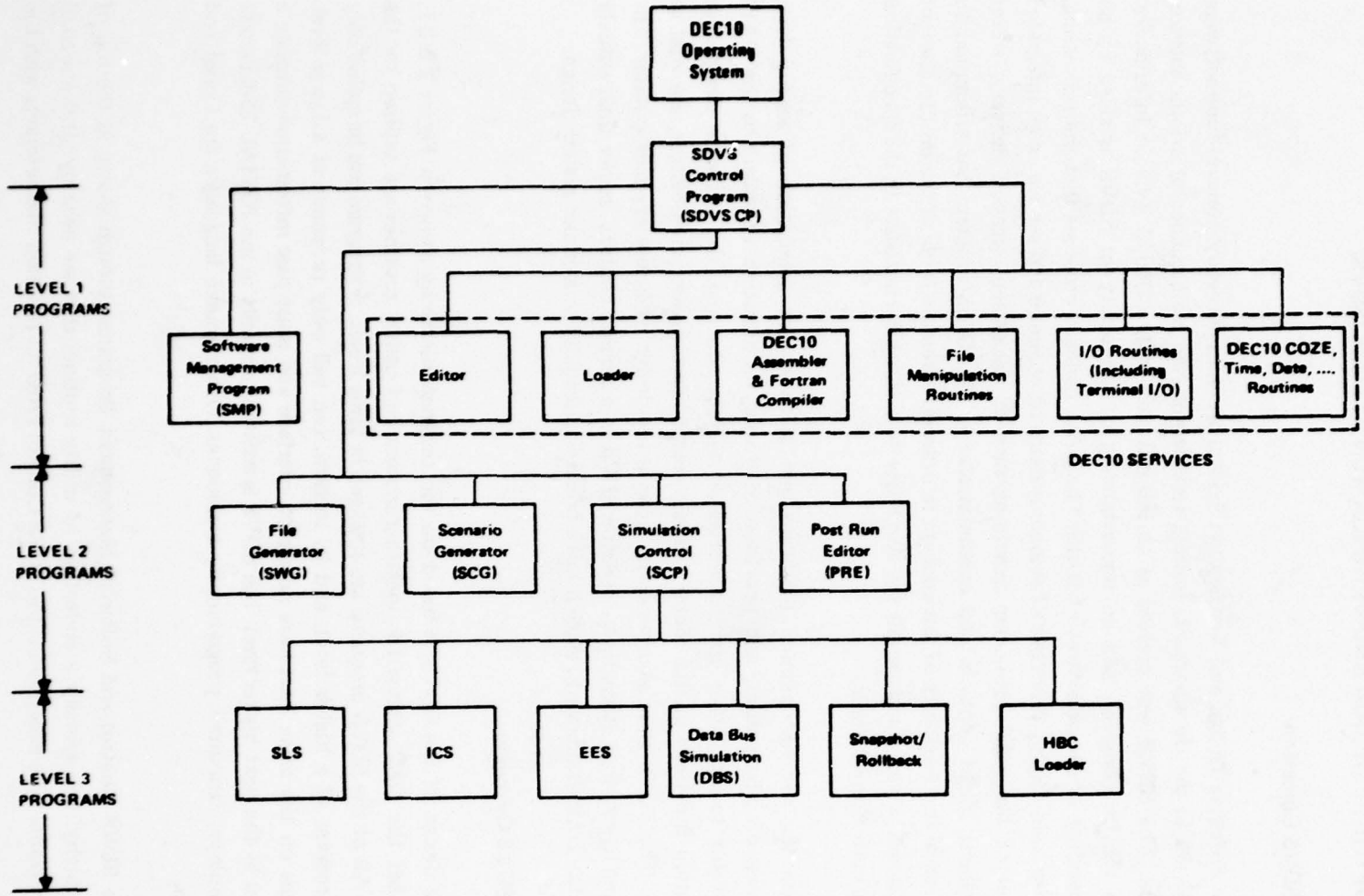


Figure 2.5.2-1. Hierarchy of SDVS software.

user and selects the mode of operation. The hierarchy diagram, as drawn, represents a functional organization rather than the exact control interfaces between the various programs. A brief statement of the function of each of the programs follows.

2.5.2.1 SDVS Control Program (CP)

This program interacts with the user in determining the desired mode of operation and then transfers control to the appropriate second level routine. All host processor functions are performed by this program (I/O handling, calling the system compiler, etc.) and are based on conversational commands input by the user from the File Generation mode and from second-level program requests. It will also submit simulation runs into the batch system.

2.5.2.2 Software Management Program (SMP)

This program maintains and provides controlled access to all SDVS files. It provides the capability to store, retrieve, interrogate, and protect these files by building and maintaining file catalogs containing information about the files. Examples of such information are user file name, internal file name, file type, program version number and revision number, creation date, security lock, and author.

To manipulate effectively the File Management data base, this program performs three major processing tasks as follows:

1. File Retrieval — Whenever a file from the File Management data base is required by one of the SDVS programs, the SMP will be invoked to retrieve the file via the SDVS Control Program. It will use the retrieval data supplied to it to interrogate its file catalogs and locate the internal name of the requested file. This internal name will be passed to the SDVS CP, which will use it to actually locate and retrieve the file from secondary storage.

2. File Disposition — When a new file is created by an SDVS program or a user, it must be placed in the File Management data base under control of the SMP. The program which generates the file (e.g., Scenario Generator, Simulation Control, etc.) issues a file creation request. This request contains the qualified name under which the file is to be controlled, the file type, and the name of the person responsible for the file. The request is passed to the SMP where the data contained in the request is used to create a new catalog entry for the file. The cataloged file is then written to secondary storage by the SDVS CP.

3. File Protection — The third major function of the SMP is to provide security locks on the files such that a given file can only be accessed by someone possessing the matching security key. Each programmer has security protection over his files until his software has been completely tested and is ready to become part of the official system. However, the project manager always has read-only access to all files in the data base.

2.5.2.3 File Generator (SWG)

This program processes file manipulation commands input by the user. The following commands illustrate some of the functions provided:

HELP	—	List the format of all SDVS user commands
ACCESS	—	Make available a specific file, version, revision for processing
EDIT	—	Perform text editing on a file
TRANSLATE	—	Compile a J73 program
COPY	—	Copy a file onto another file name
NEXT-VERSION	—	Generate a new version from the previous revision of the version specified
PRINT	—	Print a file on the system hard copy device
CREATE	—	Create a new file
ENTER	—	Enter a file from the host computer into the SDVS catalogs
OUTPUT	—	Output a file from SDVS to the host computer

The File Generator does not actually perform these functions itself (e.g., TRANSLATE, EDIT, ACCESS, etc.), but rather passes the user's requests to the SDVS Control Program which passes them to the DEC10 monitor and/or the Software Management Program.

2.5.2.4 Scenario Generator (SCG)

This program is divided into two program translators, a Simulation Control Language (SCL) translator and a Data Processing Language (DPL) translator. The SCL defines the user's simulation scenario to be executed; the DPL defines the data processing to be performed on simulation data. The SCG is executed by a conversational command to translate either an SCL or DPL file. The Scenario Generator retrieves the desired file from the SMP catalogs, translates the source code, and catalogs the translated test case in the SMP catalogs.

2.5.2.5 Simulation Control (SCP)

These programs are used to sequence a simulation scenario defined by a translated SCL program. They initialize the necessary simulators (ICS, SLS, data bus, environment) for execution, load the users mission software to be tested, perform rollbacks, and execute a simulation by invoking the various simulators.

2.5.2.6 Post Run Edit (PRE)

The Post Run Edit program provides the user with the ability to analyze the data recorded on a Rough Output Tape (ROT) during a simulation run. The Post Run Editor accesses the user-specified translated Post Run Edit directives file. The directives specify what data is to be selected from the ROT, what analysis is to be run on that data, what format is to be used to display the analysis results, what user routines are to be used, and what devices are to receive the output files created by the Post Run Editor. The Post Run Editor provides tabular printouts, interactive displays and data plots based on user directives. An important feature is the ability for a user to write an analysis routine in JOVIAL and have it execute within the framework of the Post Run Editor.

2.5.2.7 DAIS Simulators (ICS, SLS, DBS, EES)

The programs simulate the DAIS hardware. Each of these programs simulates appropriate events (e.g., execution of an instruction, a bus transmission) upon direction of the Simulation Control Program. The programs are:

ICS	—	Interpretive Computer Simulator
SLS	—	Statement Level Simulator
DBS	—	Data Bus Simulator
EES	—	External Environment Simulation

While the first three of these particular simulations have been written particularly to simulate the DIAS hardware, a brief statement of their function provides additional insight into the capability provided by SDVS. Further, the External Environment Simulation provides a simulation of avionics, airframe, and environmental conditions which interact with the software being simulated by either ICS or SLS, so that EES is not DAIS-specific, but can be utilized with any software simulation.

2.5.2.7.1 Interpretive Computer Simulator (ICS)

The purpose of the ICS is to provide a functional simulation of a computer (in this case the DAIS Hot Bench Computer) in the absence of the actual computer. The ICS is a software module of the overall SDVS system and executes mission software written for the actual computer in order to expedite debugging, integration and execution of these programs. Multiple computers can be simulated with the ICS software. The individual computers are simulated serially by the ICS, but the clock is advanced in a manner to simulate parallel operation of multiple processors.

The ICS simulates the operation of a computer at the instruction level. The instruction set is simulated in such a way that the resulting contents of registers and memory is the same as would result from actual operation of the computer. Instructions are simulated one at a time, and input/output operations are carried out with other simulation models (e.g., the EES models) after each instruction. The simulation includes operations of input/output, all addressable registers and memory. Instruction execution times are passed from the ICS to the Simulation Control Program in order for the computer execution times to be coordinated with other aspects of the simulation; control is returned to the SCP after each instruction. Figure 2.5.2.7-1 illustrates the interface of the ICS with other modules of the SDVS system.

2.5.2.7.2 Statement Level Simulation (SLS)

Whereas the Interpretive Computer Simulation simulates computer software execution at the instruction level, the Statement Level Simulation simulates software execution at the JOVIAL source statement level. The SLS interacts with the JOVIAL compiler to produce DECsystem-10 code corresponding to one JOVIAL statement at a time. Multiple SLSs may run concurrently in SDVS in order to simulate multiprocessor systems. Simulated execution time is provided by the JOVIAL compiler based on statement execution times provided by the programmer. The SLS uses these times to provide a rough synchronization between concurrently running SLSs. An illustration of the interaction of SLS with the modules of SDVS and the JOVIAL compiler is provided by Figure 2.5.2.7-2.

2.5.2.7.3 Data Bus Simulator (DBS)

The DAIS-oriented simulations described in this section allow the simulation of a multiprocessor architecture with data transfer via a multiplexed data bus. The data bus is assumed to interconnect the processor with sensors through intelligent remote terminals and

AD-A055 591 RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 1/3
AFAL SIMULATION FACILITY/CAPABILITY MANUAL. VOLUME I. EXECUTIVE--ETC(U)
JUN 77 R A WHISNANT, W H RUEDGER, R L EARP F33615-76-C-1308
UNCLASSIFIED AFAL-TR-77-118-VOL-1 NL

4 OF 5
AD
AD55591

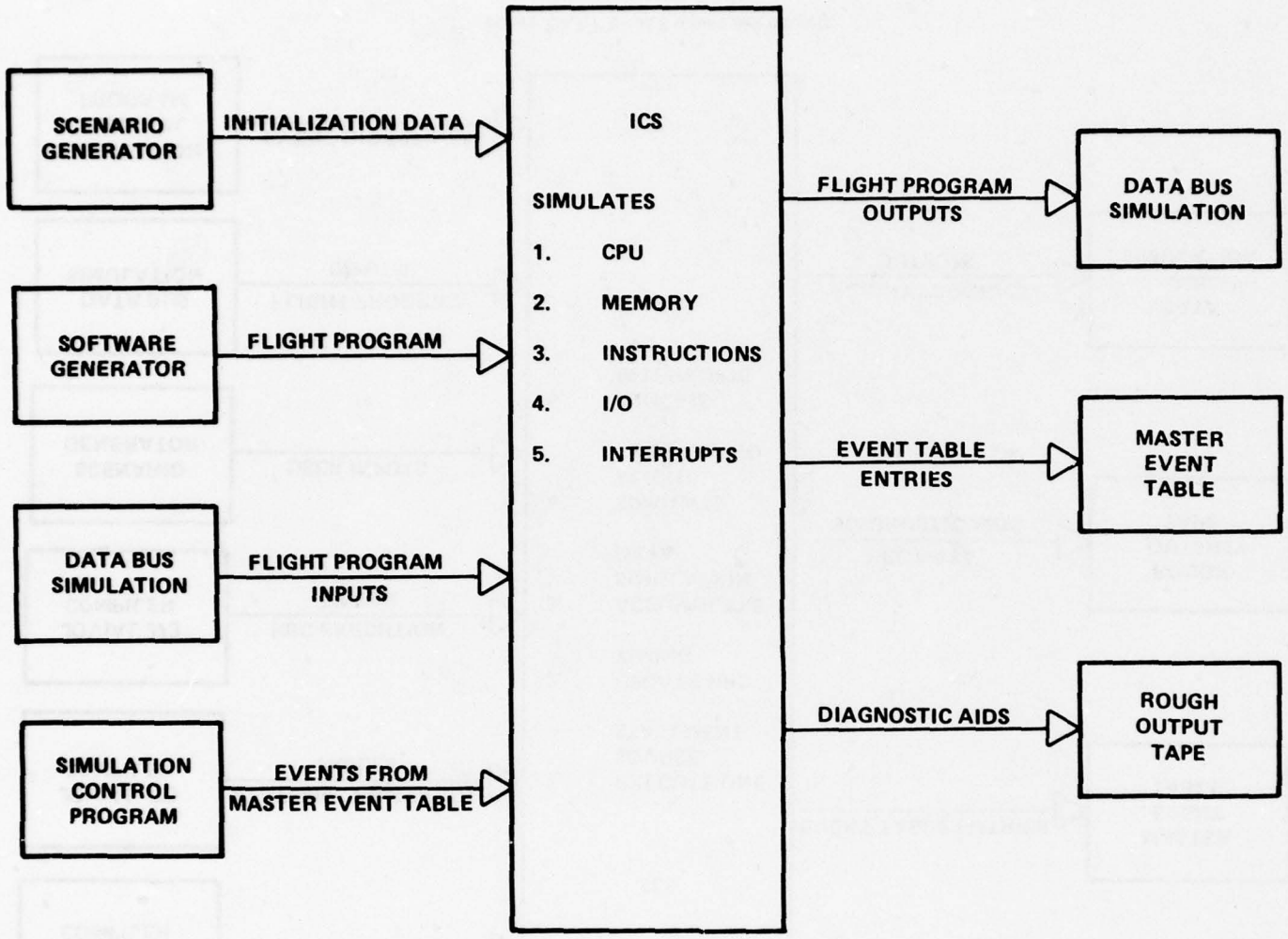


Figure 2.5.2.7-1. ICS inputs and outputs.

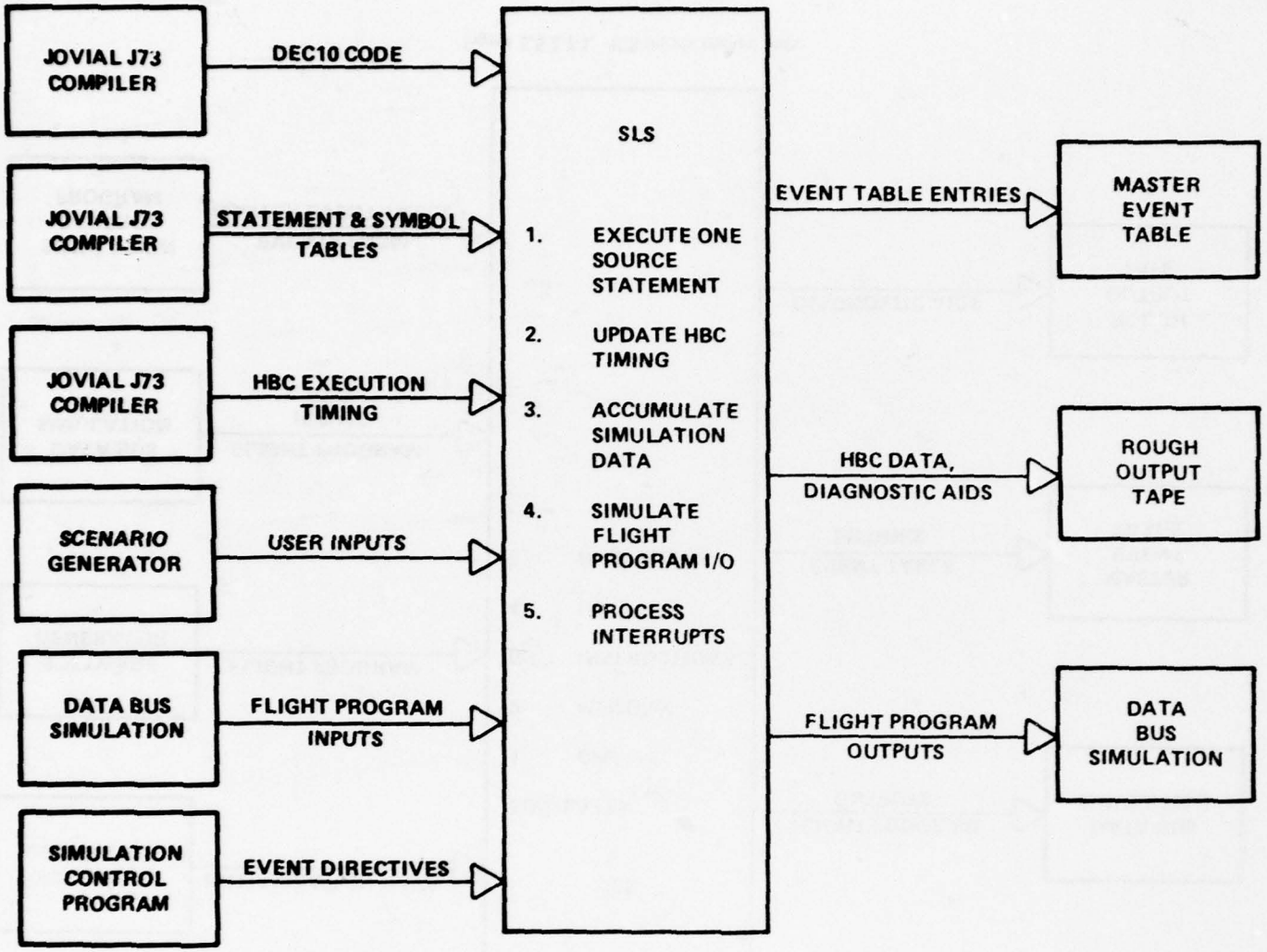


Figure 2.5.2.7-2. SLS inputs and outputs.

to conform to MIL-STD-155A bus protocol. The DBS is thus essential to a simulation of mission software interaction with the external world represented by sensors/actuators.

The DBS simulation is set up by the user, using SCL, by providing four types of information:

1. Values of SDVS variables which control the attributes of the data bus such as transfer rate and gap time,
2. Values of SDVS variables which define the remote terminals addresses and subaddresses,
3. Values of variables which effect the simulation of data bus hardware errors, and
4. Values of SDVS variables which cause the initiation of interrupt services which are associated with data bus interrupts.

These variable descriptions also suggest the functions performed by the DBS. Additional insight into the interface between DBS and the SLS, ICS, and EES is provided by Figures 2.5.2.7-1, 2, and 3 in this section.

2.5.2.7.4 External Environment Simulation (EES)

When the mission software modules being simulated are those which would normally require interaction of the software with avionics hardware (e.g., air speed data or airframe dynamics parameters), the hardware with which such interactions would take place is modeled by the External Environment Simulation. Specifically, EES provides the simulation of the environment as represented by the areas of mission profile, aircraft, and flight dynamics models, environmental models, and sensor models. The relationship of the EES to the other SDVS modules is shown in Figure 2.5.2.7-3. The EES receives inputs from the mission software being simulated via the data bus or executive simulation, all of which are being sequenced by the Simulation Control Program. The EES models compute the required data and return it to the mission software simulation. The Simulation Control Program may also record model outputs on the Rough Output Tape as directed by SCP.

The EES models, which are derived from the same basic set as those described under AVSIM (Section 2.6) can be called on either a demand or periodic basis. The EES models are listed in Table 2.5.2.7-1. Their function is to simulate aircraft position and external world state. Models may be called on a user-specified periodic basis or they may be called on a demand basis when the Data Bus Simulation makes a request to store data into or extract data from that model's variables. Similarly, models of the switches available to the pilot in

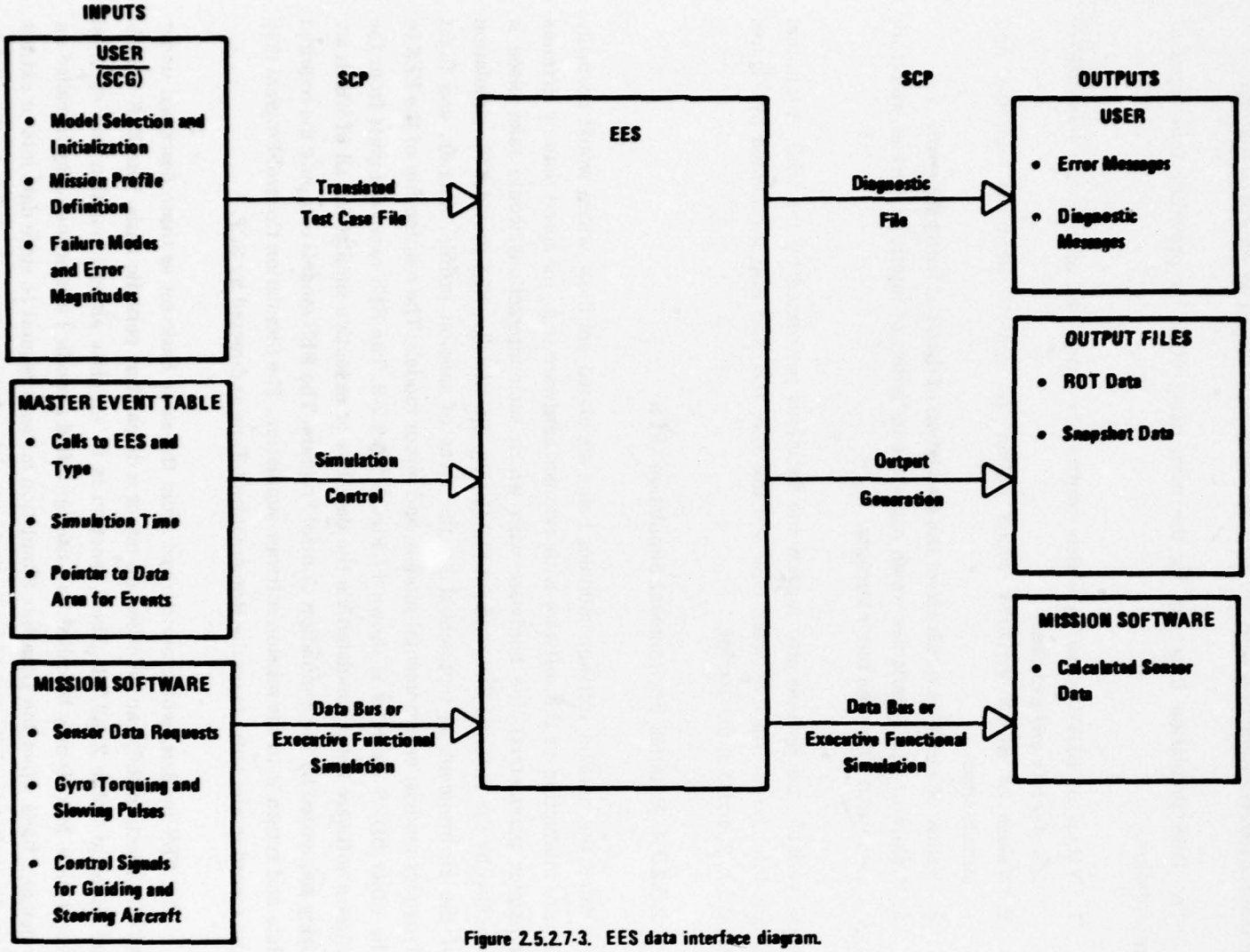


Figure 2.5.2.7-3. EES data interface diagram.

TABLE 2.5.2.7-1. EES PERIODIC MODELS

Model Name	Function of Model
Airframe (AFI)	Utilizes linear aerodynamic coefficients to model airframe dynamics in 6 degrees of freedom.
Atmosphere (ATMO)	Simulates atmospheric conditions such as temperature, pressure, and wind.
Doppler (DOPLER)	Simulates doppler radar measuring ground speed and draft angle.
Earth (EARTH)	Models rotating oblate spheroid earth to provide aircraft to earth data in flight.
Ground (GROUND)	
Inertial Measurement Set (IMS)	Models ASN-90 IMS by computing north-east-vertical incremental velocities and accepting gyro-torquing pulses.
Propel (PROPEL)	Calculates thrust of TF41-A-1 engine as function of mach number and altitude.
Air data computer (ADC)	Models free air temperature, static pressure and impact pressure as a function of altitude.

the cockpit simulator (Section 2.1.2.3) are provided in EES. Switch models are listed in Table 2.5.2.7-2. Interaction of these models with SDVS is illustrated by Figure 2.5.2.7-4. A status word indicating the status of the keyboard and a switch position is entered into common for use by software simulation programs.

The EES also provides the user with two options for specifying aircraft position data: either from a dynamic model or from a data tape. Depending on the altitude of the aircraft, either the GROUND or AIRPLANE model is called when the user chooses the model option. When the data tape option is chosen, position data is input from a tape whose name is specified in a DATA-TAPE statement. In this case, the user can input also the number of periodic cycles between update of the flight profile data from tape.

TABLE 2.5.2.7-2. EES COCKPIT CONTROL DEMAND MODELS

Model Name	Function of Component Modeled
Master Mode Panel Switches (MMPSW)	Select one of 15 mission segments (e.g., preflight, takeoff, climb, radar bombing)
Integrated Multifunction Keyboard (IMFKS)	Select one of 7 processor functions (e.g., communication, sensors)
Data Entry Keyboard Switches (DEKSW)	Keyboard entry of data (0-9, ENTER, CLEAR).
Vertical Situation Display Switches (VSDSW)	Horizontal/Vertical Situation Display Swap, 2 spares
Horizontal Situation Display Switches (HSDSW)	Select FLIR, Radar, Range up/down, Track up/down, chart
Multipurpose Display One Switches (MPD1P)	Select vertical/multipurpose/horizontal/other display
Multipurpose Display Two Switches (MPD2P)	Select vertical/multipurpose/horizontal/other display
Multifunction Keyboard (MFK)	Select one of 7 processor functions (e.g., communications, sensors)
Processor Control Panel Switches (PCPSW)	"Reconfigure" or "start" processor
Armament Panel Switches (APSWS)	Select jettison, fusing, and guns high/low
Sensor Control Unit Switches (SCUSW)	Select FLIR, sorters, laser, radar, etc.
Flight Control Stick Switches (FCSSW)	Control trigger, armament release, target-weapon, pitch/roll trim, etc.
Left Console Panel 2 Switches (LCPES)	Select flight control and electrical power functions
Left Console Panel 10 Switches (LCP0S)	Air-air initiate and air ignite switches

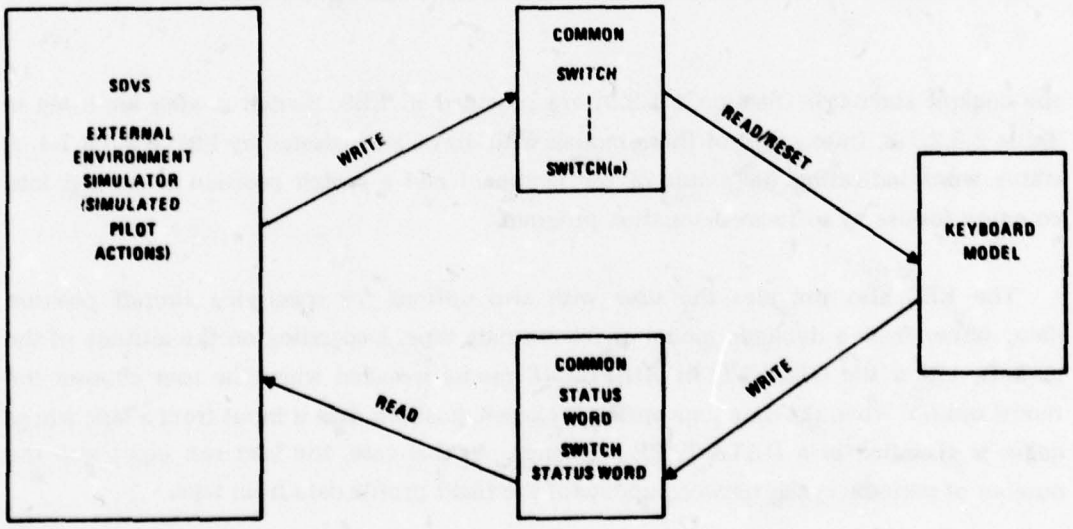


Figure 2.5.2.7-4. SDVS/keyboard model interaction.

2.5.2.8 Snapshot/Rollback

The Simulation Control Language includes a SNAPSHOT statement that, when executed during the course of a simulation, results in saving the state of the mission software, the code executors, the data bus model, and the environment models. A snapshot can be performed based on a conditional event (e.g., WHEN A > 50 or B < 30 THEN PERFORM SNAPSHOT), or at periodic intervals as defined in the Simulation Control Language.

The user can later initiate a rollback via the SDVS Rollback mode to any simulation snapshot point and restart the simulation from that point. In restarting a simulation, the user can modify his test case files to delete or add new conditions to be evaluated, reinitialize mission software and environment model data, and add or delete SCL control blocks (subroutines).

2.5.2.9 Hot Bench Computer Loaders (HBCL)

The HBCL performs the task of loading J73 and HBC Cross Assembler object files into one or more simulated computers. The user specifies the files, and how they are to be loaded, through the SCL CONFIGURE statement (Section 2.5.3.2). The resulting load map is produced in the simulation run's log file. Also, the user can request that the Loader generate an ASCII file, readable by the Hot Bench Computer Bootstrap Loader, which describes the core image loaded for an ICS simulation run.

The SCL CONFIGURE statement allows the SDVS user to specify the type of simulation (either SLS or ICS) and the object files to be loaded and executed on one or more simulated computers. The object files may be mission software procedural (both executive and application modules), COMPOOL, and/or uncataloged files. For cataloged procedural files, the SCL Compiler will automatically configure any COMPOOLS referenced by that file which have not previously been configured. The list of REL files to be loaded on each simulated computer (the File Directory) will not contain duplicate files, except for library files. A library file can be specified more than once on a single computer to resolve undefined external references which exist at that point in the load.

Also, through the CONFIGURE statement, the SDVS user can optionally specify the starting load addresses for each file's data and code segments for an ICS load. This feature does not apply for SLS loads.

2.5.3 Using SDVS

2.5.3.1 Modes of Operation

The various programs described in Section 2.5.2 are invoked by entering one of the seven basic modes of the SDVS. These modes may be executed in either conversational or batch format depending on the user's needs. The desired mode is selected upon entry to the SDVS by typing "R SDVS" as illustrated below.

```
R SDVS
WELCOME TO SDVS VER.3B(760611), YEAR, MONTH, DAY
  (LOGS NAMES L14370 ASSIGNED TO THIS SDVS RUN)
SDVS IS READY. WHICH MODE OF OPERATION IS DESIRED?
+++HELP
PLEASE ENTER NAME OR INITIALS OF ONE OF THE FOLLOWING:
    FILE GENERATION                    (FG)
    SET UP & RUN SIMULATION            (SURS)
    POST RUN EDIT                      (PRE)
    ROLLBACK                           (RB)
    DELETE MODE (MANAGER ONLY)         (DM)
    SUPERVISOR MODE (MANAGER ONLY)     (SM)
    LOGOFF                             (LOG)
```

Based on the user reply, one of the modes will be entered by SDVS and the desired operations can be performed.

2.5.3.1.1 File generation mode

The File Generation mode provides the necessary tools and configuration management aids for maintenance of all files associated with the development, test, and verification of software. An extensive cataloging system is maintained for a number of different types of software controlled by SDVS including; mission software, SDVS test case files (defining simulation scenarios and data collection requirements), environment and aircraft models, and post simulation data reduction and analysis programs. Manipulation of files cataloged in SDVS is provided for by a number of conversational commands (e.g., editing, compiling, printing, etc.) listed in Section 2.5.2.3

2.5.3.1.1.1 File Structure. Each file type is cataloged by SDVS on a version/revision

basis. For example, when the user creates a mission software file, it is cataloged as version 1, revision 0 and stored in a "baseline file". As the user edits the file in later sessions, he creates a number of revisions. Each revision results in the edited changes being cataloged as a unique record in the "difference file" for the particular file version. At any point, he may combine all or part of the revisions associated with a particular version and make a new version with the conversational NEXT-VERSION command. Under SDVS the user can access any version and revision number for a file since each EDIT session generates a unique entry in the difference file for a particular file version. In interpreting conversational commands, SDVS will interrogate the Configuration Management Catalogs to determine if the user has authority to access the desired file as established under the supervisor mode.

2.5.3.1.1.2 File Types. There are three basic file types maintained by SDVS: mission software, simulation test case, and post run edit. Several other file types are described by the SDVS User's Manual. Each of the three basic types is described briefly in the following.

The SDVS catalogs provide configuration control for the development, test, and maintenance of the mission software. The user has the capability to create and edit JOVIAL code and COMPOOL data files. SDVS, in response users commands, will automatically link COMPOOL data files to program files in the catalogs. The user is able to produce listings, save newly created and updated files, and invoke the JOVIAL J73 compiler or the DAIS processor assembler. The SDVS will automatically catalog all revisions made to a mission software file, and catalog object modules from successful compilations.

The File Generation mode of SDVS operation also provides the user the capability to create, modify, and translate source test case files containing Simulation Control Language (see Section 2.5.3.2) statements. These source test case files provide the directives, which define the initialization and control of a simulation run including sequences of operations, failure conditions, outputs to the rough output tape for post processing in the Post Run Edit Mode, etc.

The user inputs to build test case files are of two types, Conversational Language and Simulation Control Language. The user enters Conversational Language commands to enter the appropriate file handling mode of operation, i.e., create, edit, print, copy, etc. The test case files themselves will contain statements in the Simulation Control Language which are, at user request, translated to an internal form for later use in directing a Simulation run.

The primary output of building test case files are the internal test case files which are used to control the initialization and execution of simulation runs. In addition, the user

receives interactive outputs during file manipulation, such as successful completion and error messages.

The test case directives file will reference mission software files that are to be used in the simulation. It will provide directives to generate the rough output tape which will be analyzed after the simulation run is complete. The test case directive source and internal files are maintained in the SDVS file catalogs.

The File Generation mode of SDVS operation also provides the user with the capability to create, modify, and translate source Post Run Edit directives files that will contain statements in the Data Processing Language which, at user request, are translated to an internal form for input to the Post Run Editor.

The primary output of this mode of operation are the internal PRE directives files that are used in the Post Run Editor SDVS mode to perform data editing functions on a particular rough output tape generated by a simulation run. In addition, the user will obtain interactive outputs, such as successful completion and error messages, during the file manipulation and translation process. The PRE directives files, source and internal, are cataloged by SDVS. A PRE directives file may be specified by the user in a test case file to be automatically run at the end of a simulation run.

2.5.3.1.2 Set-up and run simulation mode

This mode of operation is used to submit a simulation run based on a test case directives file that has previously been created and translated. The user inputs for this mode of operation include:

1. Specification of either disk or tape,
2. Specification of the test case file,
3. Maximum simulation time,
4. Post Run Edit Prompting commands,
5. Time of day or delay time for executing the simulation SDVS will automatically:
   a. Retrieve the test case file and load the specified mission software for simulation,
   b. Interact with the machine operator to mount the necessary tapes,
   c. Perform initialization commands specified in the user's test case,
   d. Execute the desired simulation scenario and produce a Rough Output Tape (ROT), and
   e. If specified in the test case, automatically transfer control to the Post Run Edit mode and perform post run data analysis and editing on the simulation data.

The simulation run is always performed in the batch mode even if initiated from an interactive terminal.

2.5.3.1.3 Post Run Edit mode

The Post Run Edit mode provides the user with the ability to analyze the data recorded on a Rough Output Tape during a simulation run. The Post Run Editor accesses the user-specified translated Post Run Edit directives file. The directives specify what data is to be selected from the ROT, what analysis is to be run on that data, what format is to be used to display the analysis results, what user routines are to be used, and what devices are to receive the output files created by the Post Run Editor. The Post Run Editor provides tabular printouts, interactive displays, and data plots based on user directives. An important feature is the ability for a user to write an analysis routine in JOVIAL and have it executed within the framework of the Post Run Editor.

2.5.3.1.4 Rollback mode

The rollback function provides the user with the capability to restart and rerun an SDVS simulation from a point during a previous simulation run as stored on a snapshot tape (Section 2.5.2.8). The user may change the test case to obtain additional output or alter existing conditions, following the point saved on the snapshot tape. The user does this by generating a Rollback test case file which is merged with the one used for the earlier simulation by SDVS. The user will input a specification of the Rollback test case file to be used and the original Test Case File. The outputs of this mode of operation are a simulation run that (if changes were not made in the test case file) will exactly match the previous one. Changes may be made to provide further analysis of a simulation run that is of special interest.

2.5.3.1.5 Delete mode

This mode of operation is only available to the SDVS manager and provides him with the capability to delete files from the various SDVS catalogs. This function was made a manager level function to allow manager level control over the disposition of all files.

2.5.3.1.6 Supervisor mode

This mode, like the Delete mode, is available only to the SDVS manager for configuration control purposes. One of the features of the SDVS file management scheme is

that before a user may generate any file in File Generation mode, specifications must have been created for that file, including the provision of a list of users who are authorized to write (e.g., create new versions) on the file and, if appropriate, a list of users who are free only to read it. The creation of file specifications must be performed from Supervisor mode. This mode can be entered only by an SDVS user logged in on the special Manager programmer number.

In Supervisor mode, statements in the Conversational Language will be entered. One of these statements will allow the manager to create specifications for a particular file, and enter the initial list of users who have authority to read and/or write that file. Another statement allows the manager to change the authority of a user from "read only" to "read/write" or vice versa, or add new users to the list of authorized users, or remove users from the list. Both of these commands may be completely specified by the user or he may elect to be partially or entirely prompted for the necessary information.

There is only one type of output to the authorized user from Supervisor mode. This output consists of information relayed to the user about the result of processing his request. This might be a description of any syntax error that has been detected by SDVS or an indication that an error occurred while the request was being processed, or a message indicating that the request was satisfied. The specification files and the lists of authorized users are inaccessible to any SDVS user whether in Supervisor mode or not.

2.5.3.1.7 Logoff mode

After exit from any of the above modes of SDVS operation, the user will be prompted to select a mode of SDVS until the user selects LOGOFF. Upon selecting SDVS LOGOFF, the user is queried as to whether he desires the transaction log to be printed, except that the log is always printed in batch mode.

2.5.3.2 SDVS User Languages

The File Generation mode of SDVS utilizes two special languages, Simulation Control Language (SCL) and Data Processing Language (DPL) to provide SDVS user a simple and effective means of structuring simulations and obtaining results of those simulations. A major function of SDVS, in addition to providing a tool for management of software development, is to implement software simulations. The limited descriptions and examples of SCL and DPL, which follow, are important to an appreciation of the nature of simulation with SDVS.

2.5.3.2.1 Simulation Control Language (SCL)

The SCL is a programming language. Programs written in this language are compiled by the Scenario Generator (Figure 2.5.2-1), loaded by the Hot Bench Computer Loader and interpretively executed by the Simulation Control Program (SCP). The SCL provides the mechanism for the user to control a simulation by specifying initial conditions. scheduling events to occur based on time or conditions, and specifying output to be generated.

The SCL language syntax is patterned after the JOVIAL language and can be categorized into (1) non-executable statements, (2) sequential statements, and (3) asynchronous statements. The non-executable statements are used to convey control information to the simulation system such as:

1. The mission software to be simulated,
2. The flight profile tape to be used,
3. The Post Run Edit program to be executed after the simulation,
4. The variables to be traced,
5. The type of computer simulation (ICS or SLS), and
6. The type of rollback (and time).

Certain sequentially executed statements provide many of the capabilities of conventional programming languages such as FORTRAN, PL-1, and JOVIAL. Other sequential statements direct the Simulation Control Program to perform a simulation-related function. These statements are used to:

1. Assign values to variables,
2. Transfer control to other statements,
3. Evaluate a logical expression and execute one of two statements depending on the value of the expression,
4. Activate simulated computers,
5. Collect data,
6. Turn traces on and off, and
7. Terminate the simulation.

Asynchronous statements are not executed sequentially; they are executed asynchronously as the result of a user-specified condition becoming true. In a sense, the true state of the condition behaves as a software interrupt that triggers the execution of the statement.

In anticipation of an example of the use of SCL, a brief explanation of the basic structure of an SCL simulation and a cursory description of the SCL commands is in order. A simulation in SCL is configured either as an "initial test case," in which the SCL statements are structured into three distinct sections (each of which is optional), or as a "rollback test case" with two sections. The discussion here will be limited to the "initial test case." Those sections which appear in the initial test case must do so in this order:

1. Configure section,
2. Block section, and
3. Time zero section.

The configure section contains a description of the Mission Software and SDVS files that are to be loaded for the simulation. The block section contains the definition of all blocks that are to be referenced, The time-zero section contains statements that initialize variables, provide simulation timing information or control data recording and analysis.

2.5.3.2.1.1 Configure Section. The configure section consists of one or more CONFIGURE or INCLUDE statements. It allows the user to specify the type of simulation (either SLS or ICS) and the MSW modules to be loaded and executed. The simulation can be configured in one of two modes of operation, either "standalone" or "full-blown." In the standalone mode, the user may specify MSW application modules to be executed on either one SLS or one ICS code processor. In the full-blown mode, MSW Executive/Application modules may be specified for a maximum of 15 Hot Bench Computers (all SLS's or ICS's). The same MSW module can be specified for loading onto more than one code processor. For example, the command CONFIGURE ASSIGN 1 MSW-1/1/3 specifies a standalone mode statement level simulation of an MSW module 1/1/3 (file/version/revision). The statement

CONFIGURE PROCESSOR/ICS

specifies the Interpretive Computer Simulator. Assignment of MSW modules to a processor is accomplished by a CONFIGURE ASSIGN statement.

2.5.3.2.1.2 Block Section. The block section is made up of either block section statements or INCLUDE statements. There are five kinds of blocks, each bounded by BEGIN and END delimiters:

1. Control block,
2. Repetition block,

3. EFS block,
4. ROT block, and
5. Maneuver block.

Control block statements may be divided into four groups:

1. Unconditional statements,
2. Conditional statements,
3. IN statement, and
4. PERFORM statement.

The unconditional statements are SETVALVE, SNAPSHOT, TERMINATE and a limited PERFORM statement. The conditional statements are WHEN, WHENEVER, WHILE, IF and a limited IF used only in a repetition control block or EFS block. The WHEN statement contains a condition and a substatement. The first time the condition become true, the substatement is executed. The WHENEVER statement also contains a condition and a substatement. Everytime the condition becomes true, the substatement is executed. The WHILE statement contains a condition, a repetition frequency and the name of an SCL procedure. Until the condition becomes false, the SCL procedure is performed repeatedly at that specified rate. The PERFORM statement schedules execution of a specified block either once, for the current time, or repeatedly, starting with the current *time until a specified time.*

A repetition block is a control block that is restricted to certain kinds of statements. For example, it must be used instead of a control block when executing a complex PERFORM statement or a PERFORM statement that is the dependent statement of a WHENEVER or WHILE statement. The EFS block contains exactly the same kinds of statements as repetition control block, but EFS blocks are meaningful only in standalone mode.

A ROT block consists of a list of one or more variable names separated by commas. For example,

```
BLOCK ROT-EXAMPLE
BEGIN
S:I01,I02,I03
END;
```

defines variables I01, I02, I03 whose values are to be recorded on the ROT.

The maneuver block consists of a sequence of one or more assignments to External Environment Simulator (EES) variables. For example,

```
BLOCK MAN-EXAMPLE
BEGIN
E:ALAT=0.,
E:ALONG=0.
END;
```

defines initial values for two EES variables, ALAT and ALONG.

2.5.3.2.1.3 Time-Zero Section. Three types of statements may appear in the time-zero section:

1. Initialization statements,
2. Control statements, and
3. Include statements.

The INITIALIZE statements include INITIALIZE, ACTIVATE, TRACE-LIST, DATA-TAPE and POST RUN EDIT. The INITIALIZE statement is used to assign values to SDVS, mission software and EES variables at the beginning of a simulation. The ACTIVATE statement specifies start times for the various computers defined in a simulation. The TRACE-LIST specifies those SDVS, MSW and EES variables whose values are to be traced during a simulation. The DATA-TAPE statement gives the name of a tape from which EES data is to be read during simulation. The POST RUN EDIT statement gives the name of a file of post run edit directives that are to be processed by the Post Run Editor at the completion of the simulation run.

To illustrate the use of the SCL in testing mission software, consider the development of a new navigation algorithm, NAV, that has been created and compiled in the SDVS mission software file catalogs. The user is interested in generating a flight profile such that the SDVS avionic and sensor models will generate realistic navigation sensor data for input to the NAV routine.

Figure 2.5.3.2-1 is a pictorial representation of the following flight scenario:

1. Takeoff is at latitude 35°, longitude 117° with a thrust command of 1200 pounds.
2. When the X velocity $>$ 170 fps, pitch the aircraft at 2°/second.

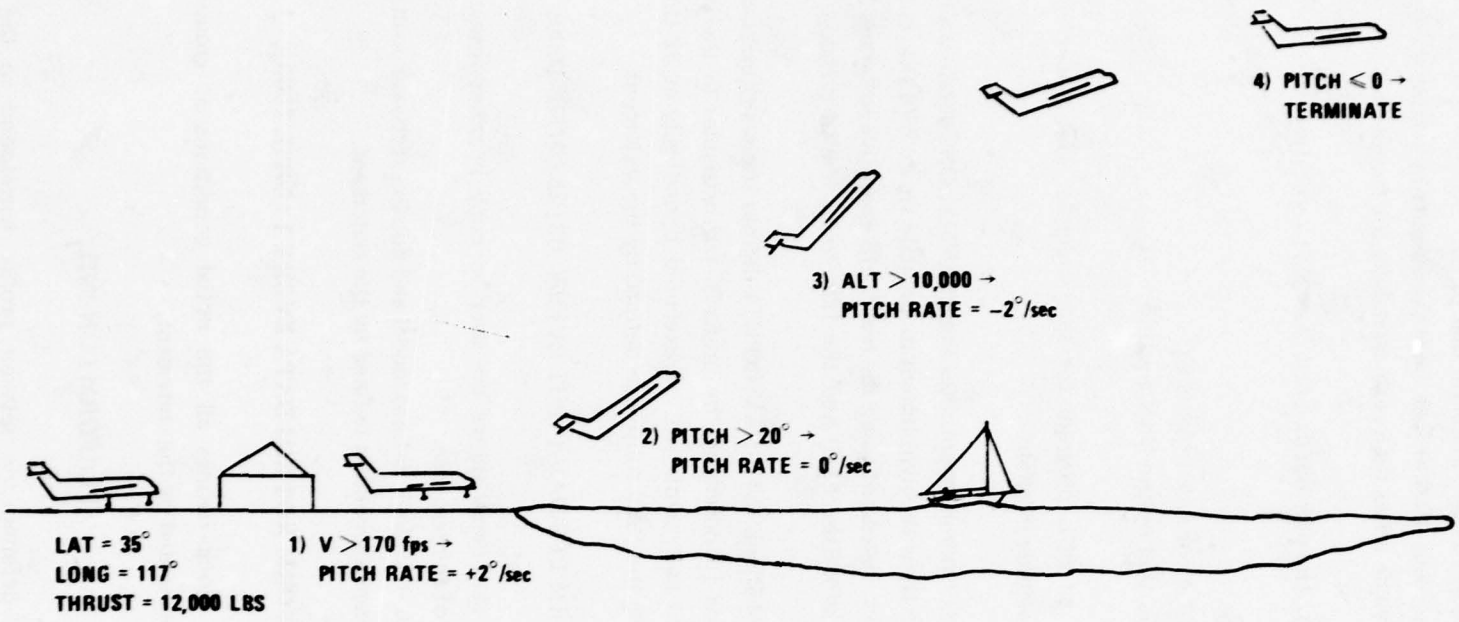


Figure 2.5.3.2-1. Sample SDVS flight profile.

3. When the pitch angle $> 20^{\circ}$, maintain that pitch.
4. When altitude exceeds 10,000 feet, level the aircraft by setting a negative pitch rate.
5. When the pitch angle is less than zero, terminate the simulation.

Using the SDVS SCL, the user builds a test case file to specify:

1. The flight profile.
2. Data to be recorded for post processing.
3. Sensor data to be used by the NAV routine.

Figure 2.5.3.2-2 is an SCL program for this example. The reader should note the following points in this sample program:

1. The CONFIGURE statement specifies the STANDALONE mode which allows a user to interface directly with environment model data via an EFS (Executive Functional Simulation) block instead of using the real DAIS executive software. It also specifies the SDVS simulator (the SLS) and the files to be loaded (version 1 revision 0 of NAV).
2. The EFS Control Block (EFS-NAV-INPUT) defines the assignment of environment mode sensor data (denoted by the prefix E:) to variables in the program, NAV. These assignment statements will be executed periodically at 32 times per second prior to executing the NAV routine as defined by the statement.

```
PERFORM EFS-NAV-INPUT EVERY .03125 UNTIL 1000;
```

3. The INCLUDE statement allows the user to copy in other test case files to be included as part of the test case.
4. The ROT-SIM-DATA block defines model and mission software position data that is to be recorded once a second as defined by the statement,

```
PERFORM ROT-SIM-DATA EVERY 1 UNTIL 1000;
```

5. The CON-INIT block defines all the initial conditions at ground zero. These assignments are executed by the statement,

```
PERFORM CON-INIT;
```

6. The statements defining the mission profile correspond to the flight profile illustrated in Figure 2.5.3.2-1.

EXAMPLE OF SDVS SIMULATION CONTROL LANGUAGE

```
"FILE DATA COLLECTION
THIS FILE DEFINES DATA TO BE COLLECTED EVERY SECOND"

BLOCK ROT-SIM-DATA
        BEGIN
            M:NAV:LAT,LONG,ALT,   "MISSION SOFTWARE POSITION"
            E:      LAT,LONG,ALT  "MODEL AIRCRAFT POSITION"
        END
```

```
"FILE GROUND-ZERO
THIS FILE CONTAINS MODEL INITIALIZATION COMMANDS"

BLOCK CON-INIT
        BEGIN
        SETVALUE E:
                ALATO   = 35/57.2957795,    "INITIAL LAT"
                ALONGO  = 117/57.2957795,   "INITIAL LONG"
                TCOM    = 12.E+3,           "THRUST COMMAND"
                QQ=0,PP=0,RR=0,             "PITCH,ROLL,YAW RATE"
                TLDF    = .10,              "10 DEG TAIL DEFLECTION"
                VU=0,VV=0,VW=0,             "BODY AXIS VELOCITIES"
                THETA=0,PHI=0,PSI=0,        "EULER ANGLES"
                ALT=0;                      "ALTITUDE"
        END
```

```
"FILE NAV-DATA
THIS FILE DEFINES THE ASSIGNMENT OF MODEL DATA TO
THE NAV ROUTINE"

BLOCK EFS-NAV-INPUT
        BEGIN
        SETVALUE M:NAV:SP  = E:STEAP,   "SINE PITCH"
                 M:NAV:CP  = E:CTEAP,   "COS PITCH"
                 M:NAV:SR  = E:SPIIIP,  "SINE ROLL"
                 M:NAV:CR  = E:CPIIIP,  "COS ROLL"
                 M:NAV:SY  = E:SPSIP,   "SIN YAW"
                 M:NAV:PY  = E:CPSIP,   "COS YAW"
                 M:NAV:VX  = E:DVII,    "INCREMENTAL X VEL"
                 M:NAV:VY  = E:DYE,     "INCREMENTAL Y VEL"
                 M:NAV:VZ  = E:DVV;     "INCREMENTAL Z VEL"
        END
```

```
"CONFIGURE THE SIMULATION IN THE STANDALONE MODE LOADING
VERSION 1 REVISION 0 OF NAV"

    CONFIGURE STANDALONE SLS NAV/1/0;

"INCLUDE TEST CASE FILES DEFINING DATA TO BE RECORDED AND
ENVIRONMENT MODEL INITIALIZATION DATA"

INCLUDE DATA-COLLECTION/1/0;
INCLUDE GROUND-ZERO/1/0;

"INCLUDE EXECUTIVE-FUNCTIONAL-SIMULATION(EFS) FILE
DEFINING SENSOR INPUT DATA THAT WOULD NORMALLY BE
INPUT WITH THE REAL EXECUTIVE AND BUS CONTROL SOFTWARE"

INCLUDE NAV-DATA/1/0;

"EXECUTE INITIALIZATION COMMANDS AT TIME ZERO"

    PERFORM CON-INIT;

"INPUT NAV SENSOR DATA AND EXECUTE THE NAV ROUTINE AT
32 TIMES PER SECOND"

    PERFORM EFS-NAV-INPUT EVERY .03125 UNTIL 1000;

"THE FOLLOWING STATEMENTS DEFINE THE MISSION PROFILE.
TAKEOFF WHEN VELOCITY > 170 FPS"

    WHEN E:UU > 170 THEN SETVALUE QQ=2/57.295795;

"MAINTAIN 20 DEGREE PITCH ATTITUDE"

    WHEN THETA > 20/57.295795 THEN SETVALUE QQ=0;

"AT 10,000 FT., LEVEL OFF"

    WHEN ALT > 10,000 THEN SETVALUE QQ= -2/57.295795;

"TERMINATE THE SIMULATION WHEN PLANE LEVELS OFF"

    WHEN QQ < 0 THEN TERMINATE;

"COLLECT POST PROCESSING DATA EVERY SECOND"

    PERFORM ROT-SIM-DATA EVERY 1 UNTIL 1000;
```

MISSION PROFILE DEFINITION

Figure 2.5.3.2-2. Sample SDVS SCL program.

2.5.3.2.2 Data Processing Language (DPL)

An SDVS simulation generates a volume of data collected at numerous points in a simulation. With the SCL, the user can specify both conditional and unconditional events that result in the output data to a Rough Output Tape. This tape contains all the simulator trace outputs, a load map of the mission software for each DAIS processor, run time error and warning messages from the various simulators, data from the environment models and mission software defined by the user, etc., as they occur in simulated time. Figure 2.5.3.2-2 illustrates the use of a Rough Output Tape (ROT) block defining the variables to be recorded every second during a simulation. From the vast volume of data, the user must be able to sort out and display the information in a meaningful format.

The SDVS Data Processing Language has been designed to provide the SDVS user with an easy-to-use, flexible tool to select for analysis, printout, or plotting the specific parameter data he desires. In selecting data to be output, the user does not have to worry about conversion of mission software or environment model data from binary to the correct output format; this is all handled automatically by SDVS. To determine the correct formats, SDVS reads the symbol tables generated for mission software and environment model programs by the JOVIAL and FORTRAN compilers, and extracts the necessary information. This SDVS tool removes much of the drudgery sometimes associated with data analysis. The DPL provides the following user oriented functions:

1. Generation and editing of data files containing user defined variables from the ROT.
2. A PRINT capability to output generated data files to the printer.
3. A DISPLAY capability to output information to the user's interactive terminal.
4. A statistical package to compute statistical information of simulation data.
5. Automatic generation of plots based on collected simulation data.
6. Execution of user supplied analysis routines using a simulation ROT.

The use of DPL is illustrated by the DPL program of Figure 2.5.3.2-3 that can be used to process data collected in the SCL example shown in Figure 2.5.3.2-2. This DPL program is used to print out the environmental model and mission software NAV data. This data will be printed on the line printer, and will be analyzed by a user routine, error analysis, to determine the mission software error. This error is then plotted as a function of time.

The reader should note the following points from this sample program:

1. The CONFIGURE statement specifies the user routine to be executed, and its language (JOVIAL).

EXAMPLE OF SDVS POST RUN PROCESSING

```
"THIS POST-RUN-EDIT PROGRAM IS USED TO PRINT OUT THE"
"ENVIRONMENTAL MODEL AND MISSION SOFTWARE NAV DATA FROM"
"A SIMULATION RUN. THIS DATA WILL BE PRINTED ON THE LINE"
"PRINTER, AND WILL BE ANALYZED BY A USER ROUTINE, ERROR"
"ANALYSIS, TO DETERMINE THE MISSION SOFTWARE ERROR. THIS"
"ERROR IS THEN PLOTTED AS A FUNCTION OF TIME"

"SPECIFY THE USER ERROR-ANALYSIS ROUTINE"

     CONFIGURE USER-ROUTINE JOVIAL ERROR-ANALYSIS/I/O;

"GENERATE THE DATA FILE, NAV-DATA, CONTAINING"
"THE PARAMETERS IN ROT BLOCK, ROT-SIM-DATA,"

     GENERATE NAV-DATA ROT-SIM-DATA;

"DEFINE THE DATA FILE CONTAINING THE OUTPUT OF THE USER"
"ROUTINE. THIS OUTPUT IS THE NAVIGATION ERROR FOR LATITUDE,"
"LONGITUDE, ALTITUDE, AND THE SIMULATION TIME, TIME"

     FORMAT NAV-ACCURACY
          BEGIN
             FLOATING:  LAT-ERR,
             FLOATING:  LOW-ERR,
             FLOATING:  ALT-ERR,
             FLOATING:  TIME
          END;

"PRINT OUT ALL THE MSW AND EES NAV DATA"

     PRINT NAV-DATA;

"EXECUTE THE USER ROUTINE, ERROR-ANALYSIS, WHICH COMPUTES"
"THE NAVIGATION ERROR"

          "INPUT FILE:    NAV-DATA"
          "OUTPUT FILE:   NAV-ACCURACY"

     EXECUTE ERROR-ANALYSIS

               NAV-DATA:NAV-ACCURACY;

"PLOT THE COMPUTED NAVIGATION ERRORS AS A FUNCTION OF TIME"

     PLOT NAV-ACCURACY  SIM-TIME, LAT-ERR(RAD-DEG)
                        TITLE='LATITUDE ERROR VS TIME'
                        XLABLE='TIME(SEC)'
                        YLABLE='LATITUDE ERROR (DEG)';

     PLOT NAV-ACCURACY  SIM-TIME, LOW-ERR(RAD-DEG)
                        TITLE='LONGITUDE ERROR VS. TIME'
                        XLABLE='TIME(SEC)'
                        YLABLE='LONGITUDE ERROR (DEG)';

     PLOT NAV-ACCURACY  SIM-TIME, ALT-ERR
                        TITLE='ALTITUDE ERROR VS. TIME'
                        XLABLE='TIME(SEC)'
                        YLABLE='ALTITUDE ERROR(FEET)';
```

Figure 2.5.3.2-3. Sample SDVS DPL program.

2. The GENERATE statement is used to create a data file, NAV-DATA, which includes all the data on the ROT block, ROT-SIM-DATA.
3. The FORMAT statement is used to define the format of the data file (NAV-ACCURACY) which is computed by the user's routine.
4. The PRINT statement will automatically print out all the data contained in the data file, NAV-DATA. The output is in a tabular format and is time tagged.
5. The PLOT commands shown allow the user to specify the plot title, the axis titles, and any desired data conversions. The user could also specify the X and Y axis lengths, the minimum and maximum X and Y values allowed, and any biases to be added or subtracted from variables to be plotted.

SECTION 2.5 BIBLIOGRAPHY

TRW Defense and Space Systems Group, "The Software Design and Verification System (SDVS)," Final Report for Period 17 June 1974 to 30 June 1976, Contract F33615-74-C-1159.

TRW Systems Group, "Software Design and Verification System, Phase III," *User's Manual*, 11 June 1976.

TRW Systems Group, "Software Design and Verification System (SDVS) Requirements Document," 15 August 1975, 6404.D-91.

2.6 AVSIM

AVSIM is a simulation facility that effects avionics system evaluation, validation, and integration by dynamic digital simulation of the airframe, flight controls, and avionics equipment of a generic high-performance tactical fighter. The objectives of this facility currently are to:

1. Test and validate operational flight programs under realistic flight conditions,
2. Effect digital avionics system integration,
3. Identify hardware/software problems in prototype avionics systems, and to
4. Recreate flight problem areas through dynamic simulation.

AVSIM is capable of simulating the navigation, penetration, and weapon delivery phases of an attack fighter mission both individually and compositely. The AVSIM user configures his aircraft, sensor complement, environmental characteristics, and target characteristics by linking individual simulation models into an overall simulator structure. The AVSIM simulator currently has real-time, non-real-time, man-in-the-loop, and self-contained modes

of operation. The user has the option of using resident (F-16) software developed by General Dynamics of Ft. Worth, using resident (A7) software obtained from the Navy, or of developing his own by utilization of resident creation routines. In the example of the resident software, the airframe is configured by selecting either an F-16 or A7 aircraft model, an appropriate flight control system dependent on desired complexity, and selecting self-contained (synthetic mission generator/simulated pilot), pre-recorded, or real-time cockpit inputs. The sensor complement presently available includes the radar altimeter, the attack radar, and (may be extended to include) electro-optical sensors. Flight environment is incorporated by utilization of models that provide simulated air data inputs, accelerometer and gyro outputs, representative weather effects, atmospheric perturbations, inertial outputs, and magnetic heading. Auxiliary software integral to the total simulation includes an inertial reference, geometry effects, and the ability to introduce noise at various points. AVSIM is hosted by the DEC-10 facility at AFAL and is linked to peripherals such as the cockpit and display generator by means of DMA bus to satellite PDP-11s. AVSIM software consists of control modules and application models. The control software provides file manipulation, sets up the simulation configuration, provides initialization, implements man-machine interface, and controls overall execution sequence. The application models provide aforementioned sensor data, external physical conditions, etc. Also contained within the software are data acquisition and analysis modules that accumulate and edit data for validation analysis.

AVSIM programs are coded largely in ANSI-FORTRAN in an attempt to make the software flexible, modular, and transferable. DEC system FORTRAN-10 features, as well as DEC-10 assembly language, are also used to a lesser degree.

The software module structure for AVSIM is shown in Figure 2.6-1. The software is organized to perform the two primary functions of set-up and execution. Subfunctions within program set-up include the use of "ENTRY.BLD" in the creation of new applications models and the use of "PRESCENARIO" to establish a new simulation configuration by creating a new "EXECUTIVE" where application model complement, sequence, and sequence rate are specified. Actual run-time initialization and parameter modification is achieved in "SCENARIO". These are discussed in greater detail in the following paragraphs. A distinction is introduced into the discussion at this point to differentiate between the executive or control software and the applications model software.

2.6.1 Executive (Control) Module Software

The executive or control modules provide the supervisory software necessary to create,

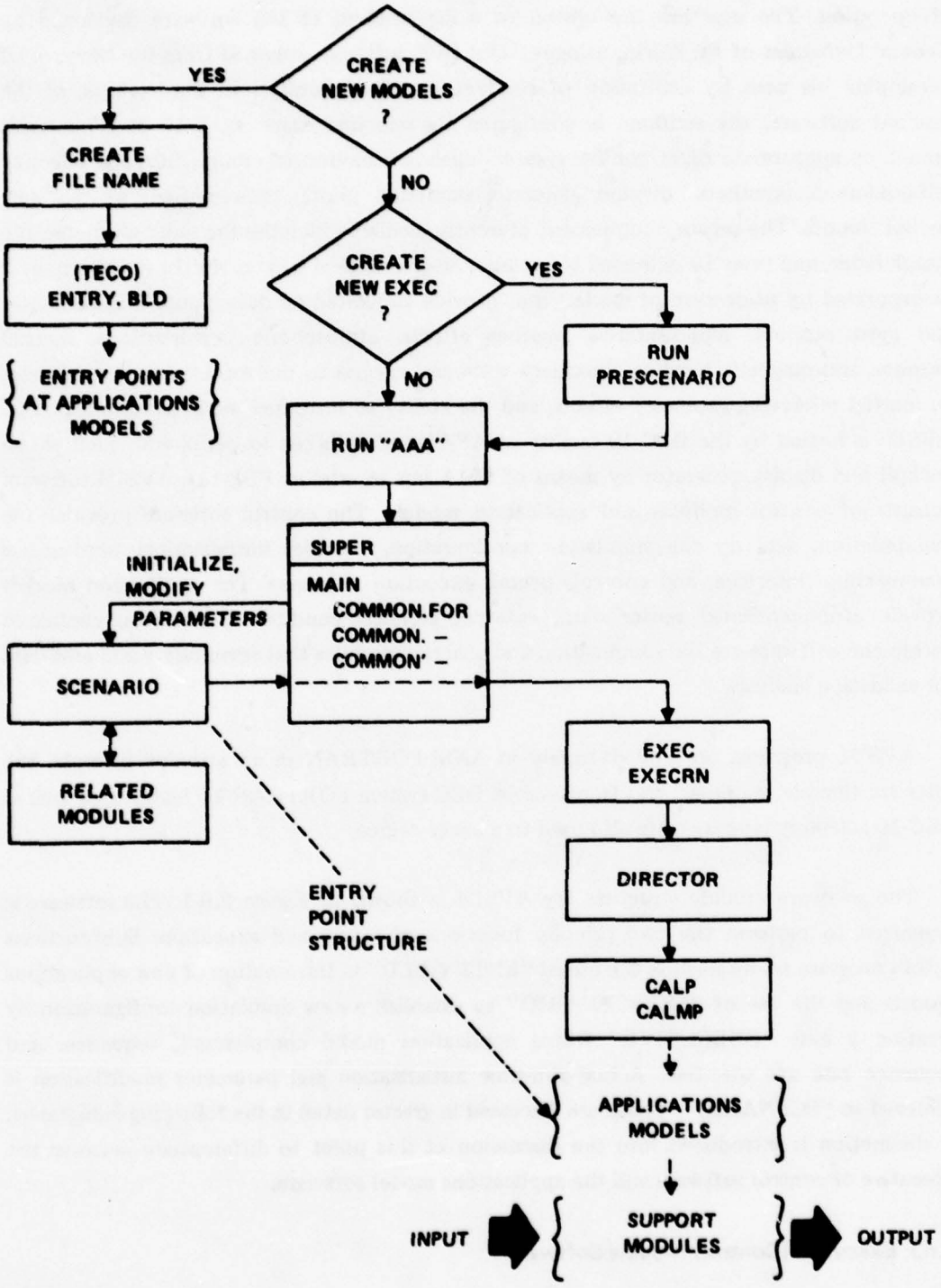


Figure 2.6-1. AVSIM simulation structure.

set up, and execute the applications software. These modules provide file manipulation and control, selection of the options to be run, initialization, operator input editing, diagnostic information, simulation monitoring, execution sequence, and execution control.

Of particular interest to the user who is creating a new simulation configuration are the routines "ENTRY.BLD" and "PRESCENARIO". As described below these provide capability to structure the entry point feature in new applications models and to configure a model set by generating a new "EXEC", respectively. The two modules of principle interest to the user at run-time are the "DIRECTOR" and the "SCENARIO". These provide man-machine interface during program execution and set-up, respectively. These and other executive and support software are described in the following sections.

2.6.1.1 Multiple Entry Points and Adding New Applications Models

Each model makes use of the FORTRAN-10 feature to link into a subroutine at a predetermined position in the subroutine source code. These entry points are addressed by the name of the appropriate subroutine to which either "IN", "TT", or "EX" have been appended. These represent source code modules within the last subroutine, which provide for "initialization" (i.e., default specification) "teletype" (i.e., operator update of the default values if desired), and "execute" (i.e., setting the actual run parameters to either the specified default value or to the updated value). For example, a link to FCSIN would enter subroutine FCS at an appropriate place to access the prespecified default parameter list. A subsequent link to FCSEX would set the run parameter to the accessed default value. These entry points are created at the same time a new application model is created through the use of ENTRY.BLD as described in the Computer Programmer's Manual.

Additionally, where a new applications model is created, submodules INCALL, INEX, EX, MODELS, and INDUM must be revised. These modules essentially exercise the model set(s) through the entry point linkages discussed above. For example, INCALL activates the initialization "IN" programs for the model set sequentially. Thus, calls to the new model must be included for each of these subprograms. In some instances it is possible to disturb execution or initialization logic if the call is improperly placed with the source code.

2.6.1.2 PRESCENARIO

PRESCENARIO is a simple routine used to configure a simulation by constructing new, or modifying existing, EXEC subroutines (i.e., create or modify the EXTERNAL statement

defining the model set). By the use of seven commands which can be entered at any remote terminal, the user can:

1. Configure new simulations (new EXEC),
2. Examine existing configurations,
3. Delete configurations no longer needed, and
4. Start a simulation using any one of the existing configurations.

The commands and their functions are briefly:

1. CREATE – creates a new executive with the name EXECn.FOR.
2. DIRECTORY – lists any/all executives and their status.
3. EXECUTE – compiles, loads, and executes simulation with the new EXEC. The name of the EXEC is specified in the argument list.
4. HELP – list all commands available and their syntax.
5. KILL – exits PRESCENARIO, stores all system information. Does not execute simulation.
6. RUBOUT – deletes any/all executives specified as arguments.
7. TYPE – types out all models and their frame rates for any/all executives.

The Programmer's Manual should be consulted regarding command mnemonic and argument(s).

PRESCENARIO is entered with the command "RUN PRESCN." After system status flags, the system response "***COMMAND!***" is returned. The user then selects the command appropriate to his objective and enters it. The logic in prescenario is very simple and consists of:

```
***** BEGIN
*          GET A COMMAND FROM USER
*          DETERMINE WHICH COMMAND WAS GIVEN
*          CALL THE PROPER COMMAND ROUTINE
*          IF NOT A KILL, GO TO BEGIN
****       RETURN
```

The General Dynamics software documentation indicates that PRESCENARIO possesses a tutor mode selectable by the user at set-up time. However, RTI review of available source code did not immediately evidence this capability.

2.6.1.3 MAIN

This module has two primary functions. The first is the establishment of program data trafficking by means of extensive common block definition. The second is the call to "SCENARIO" or "DIRECTOR" modules. A lesser function performed by "MAIN" is the initialization of I/O files.

2.6.1.4 SCENARIO

The SCENARIO provides the non-real-time interface for man-machine communication during the initialization stage of a simulator run. The operator utilizes the SCENARIO model to load, initialize, and prepare the simulator for operation. SCENARIO enables the operator to select a SCENARIO/sensor/subsystem configuration from the AVSIM library based on a predetermined configuration (in the EXEC module). The operator has the option of using stored (default) parameters or of inputting a parameter set of his own. Large scale parameter/model changes are generally handled other than at set-up by program modification. SCENARIO makes extensive use of the multiple entry feature of the DEC *FORTRAN-10 (see preceding section)* on adding new applications models for detail).

SCENARIO also provides the operator with the ability to double check and verify his input. Comments are provided for many anticipated operator mistakes as well as for assistance in setting up the simulator run and developing the SCENARIO. If a given option is selected, information defining those models which are required to implement that option is provided to the operator by means of an optional tutor mode.

The simulator operator can expect to be required to advise SCENARIO of the following input options:

1. Monitor Console Operator Mode (Tutor or Set-Up),
2. Airframe Mode (Self-Contained or Man-In-Loop),
3. Subsystem Information,
4. Sensor Options,
5. Hardware,
6. Target Options,
7. Weather Options,
8. Parameter Options, or
9. Perform Housekeeping Functions.

Output from the SCENARIO goes to the director, sensor models, subsystem models, and I/O sub-program data initialization file. DIRECTOR tables which provide for proper execution of the loaded models are set up from SCENARIO output. The DIRECTOR then interfaces with SIMSUB and CALPM in order to be synchronized with the DEC-10 real time clock or the DAIS flight processor.

2.6.1.5 EXEC

The primary function of this module is to set up the (real time) model suite. This is accomplished by defining those elements desired in an External statement. Generally, the models are selected from the AVSIM library (currently F-16 or A7 representations) but as stated previously may be user input. This module also initializes configuration parameters.

2.6.1.6 DIRECTOR/CALIPER

The DIRECTOR is the overall simulation monitor and provides the interface between SCENARIO (i.e., program set-up) and the DEC-10 Real-Time Operating System (i.e., program execution). Monitoring tasks include scheduling the execution of sensors, subsystem models, I/O interfaces, data acquisition, and analysis models. Through coordination with SCENARIO, task and priority tables are constructed that provide for the simulator to execute the proper models in the proper sequence. The user communicates with the DIRECTOR to provide control. This includes initialization, hold, or reset, etc. DIRECTOR allows the user to specify the following simulation parameters:

1. Sense switch to halt the simulator,
2. Sense switch to allow real-time analysis,
3. Run time limit,
4. Frame time (time for 64 iterations to execute),
5. Total number of frames, and
6. DMA number the simulation is synchronized to.

The DIRECTOR outputs include task priorities, special event timing, etc., to the DEC-10 Real-Time Executive. Tracking information is sent to the CRT display such that the operator is aware of progress and intermediate results. Traceback messages are extensive in non-real-time simulations with sense switches on.

The DIRECTOR interfaces with the real-time clock through assembly language program SIMGO and employs a time slice concept for sequencing models as implemented in subroutine CALP.

2.6.2 Simulation Model Set Description

Generally, a given simulation is configured by selecting appropriate applications model libraries representative of either F-16 or A7 tactical fighter scenarios. As discussed previously, alternate applications models may be user supplied, providing proper program interface is maintained. This section delineates model sets currently available and includes individual model descriptions for the F-16 library developed by General Dynamics. The discussion is intentionally brief and the user is referred to the references for more detail.

The F-16 model set developed for AFAL by General Dynamics includes the following resident modules:

Airframe,
Flight Control System,
Air Data Computer,
Accelerometers and Gyros,
Simulated Pilot,
Synthetic Mission Generator Model,
Target Simulator,
Attack Radar,
Radar Altimeter,
Random Noise and Error Generator,
Relative Geometry,
Weather,
Atmosphere,
Reference Model for INS Control,
Inertial Reference Unit, and
Flux Valve.

Figure 2.6.2-1 depicts the major paths for transferring data between the various models. The A7 model acquired from the Naval Weapons Center at China Lake, California, has been adapted for AVSIM. Approximately 23 modules are resident on the DEC-10 host. Of these, 13 have been selected by AFAL to represent the DAIS simulation scenario. These are:

Airframe,
Aerodynamic Model,
Propulsion,
Earth,

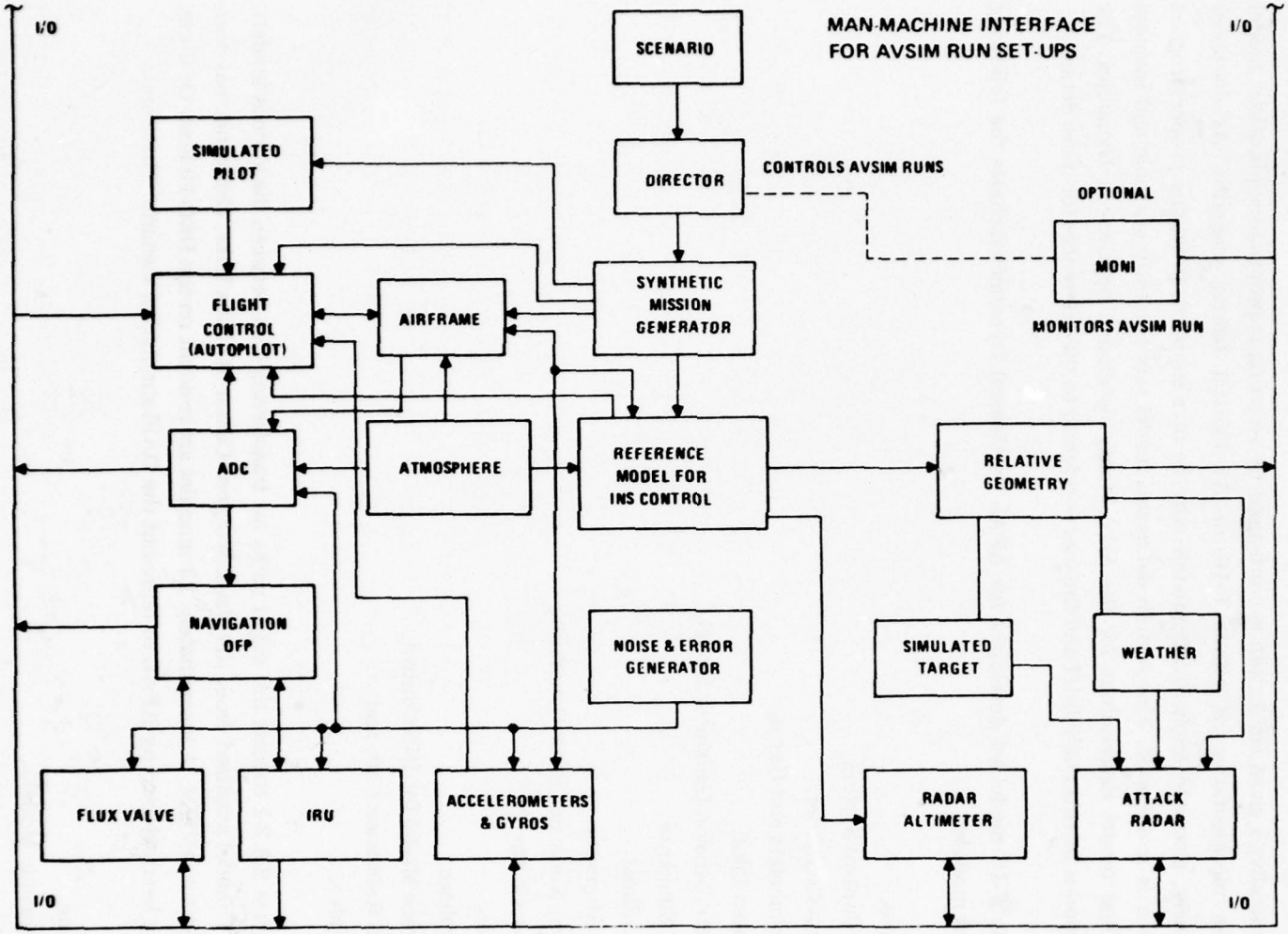


Figure 2.6.2-1. Major data paths.

Inertial Measurement System,
Atmosphere,
Target,
Radar Altimeter,
Forward Looking Radar,
Air Data,
Doppler, and
TACAN.

2.6.2.1 Airframe (AFM1)

The Airframe model provides the capability to evaluate the effects of airframe dynamics on system operation. The model utilizes linear aerodynamic coefficients to provide vehicle dynamics for six degrees of freedom. It incorporates Mach number variations of the aerodynamic coefficients and Mach and altitude variations of the flexibility factors. Variable gross weight and moments of inertia can be included through fuel flow integration and store configuration status. Figure 2.6.2.1-1 is an Input/Output Block Diagram of AFMI with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.1-1 below.

2.6.2.2 Flight Control System Model (FCS)

FCS is the mathematical model of the flight control system electronics, which includes augmentation (CAS), stability augmentation (SAS), and a command following autopilot. The CAS and SAS systems are provided to augment the control and stability of the basic airframe. The command following autopilot accepts commands in attitude and altitude for vehicle control when the cockpit simulator is not used. FCS also contains a simulation of the mechanical control surface actuators.

The major functions and/or procedures contained in the FCS subprogram are:

1. Stability and Control Augmentation, and
2. Command Following Autopilot.

The stability and control augmentation function provides the compensation networks and feedback paths to improve vehicle control and to provide the proper stability characteristics. Pitch rate, normal acceleration, and angle of attack provide the feedback in the longitudinal

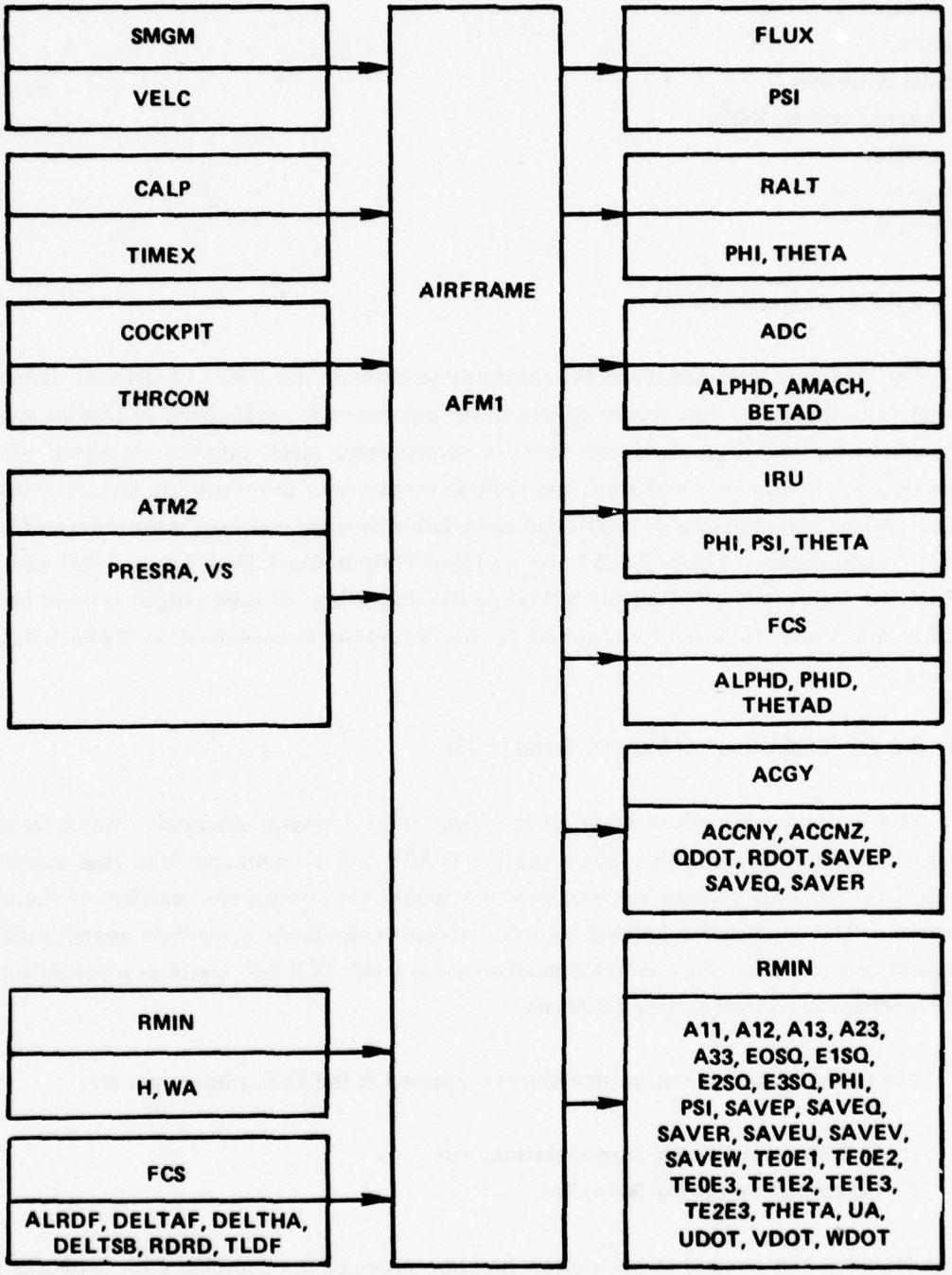


Figure 2.6.2.1-1. AFM1 input/output block diagram.

TABLE 2.6.2.1-1. AFM1 INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
Input Data			
ALRDF	δ_a	Aileron deflection	degree
DELTAF	δ_F	Flap deflection	degree
DELTHA	δ_{Ha}	Differential horizontal tail deflection	degree
DELTSB	δ_{sB}	Speedbrake deflection	degree
G	g	Earth gravity	ft/s^2
H	h	Airplane altitude (double precision)	ft
PRESRA	P/Po	Air pressure ratio	
RDRD	δ_r	Rudder deflection	degree
THRCON	–	Throttle input from cockpit	
TIMEX	t	Elapsed time since start of simulation run	s
TLDF	δ_H	Horizontal tail deflection	degree
VELC	V_C	Airplane velocity command	ft/s
VS	a	Speed of sound	ft/s
WA	–	Wander angle	degree
Output Data			
A11	a_{11}	Direction cosine matrix element	
A12	a_{12}	Direction cosine matrix element	
A13	a_{13}	Direction cosine matrix element	
A23	a_{23}	Direction cosine matrix element	
A33	a_{33}	Direction cosine matrix element	
ACCNY	$A_{Y_{CG}}$	Lateral acceleration at the C. G.	g
ACCNZ	$A_{N_{CG}}$	Normal acceleration at the C. G.	g
ALPHD	α	Angle of attack	degree
AMACH	M	Mach no.	
BETAD	β	Sideslip angle	degree
EOSQ	$(E_0)^2$	E_0 squared	
E1SQ	$(E_1)^2$	E_1 squared	
E2SQ	$(E_2)^2$	E_2 squared	
E3SQ	$(E_3)^2$	E_3 squared	
PHI	ϕ	Airplane roll angle	rad
PHID	ϕ	Airplane roll angle	degree
PSI	ψ	Airplane yaw angle	rad
QDOT	$\dot{q}$	First time derivative of q	rad/s^2
RDOT	$\dot{r}$	First time derivative of r	rad/s^2
SAVEP	p	Body axis roll rate	rad/s
SAVEQ	q	Body axis pitch rate	rad/s
SAVER	r	Body axis yaw rate	rad/s
SAVEU	u	X axis airplane velocity WRT earth	ft/s
SAVEV	v	Y axis airplane velocity WRT earth	ft/s
SAVEW	w	Z axis airplane velocity WRT earth	ft/s
TE0E1		Two x E0 x E1	
TE0E2		Two x E0 x E2	

TABLE 2.6.2.1-1. AFM1 INPUT/OUTPUT NOMENCLATURE (con.)

Output Data			
FORTRAN Name	**Math Symbol**	**Description**	**Units**
TE0E3		Two x E0 x E3	
TE1E2		Two x E1 x E2	
TE1E3		Two x E1 x E3	
TE2E3		Two x E2 x E3	
THETA	θ	Airplane pitch angle	rad
THETAD	θ	Airplane pitch angle	degree
UDOT	$\dot{u}$	X axis airplane acceleration WRT earth	ft/s^2
VDOT	$\dot{v}$	Y axis airplane acceleration WRT earth	ft/s^2
WDOT	$\dot{w}$	Z axis airplane acceleration WRT earth	ft/s^2

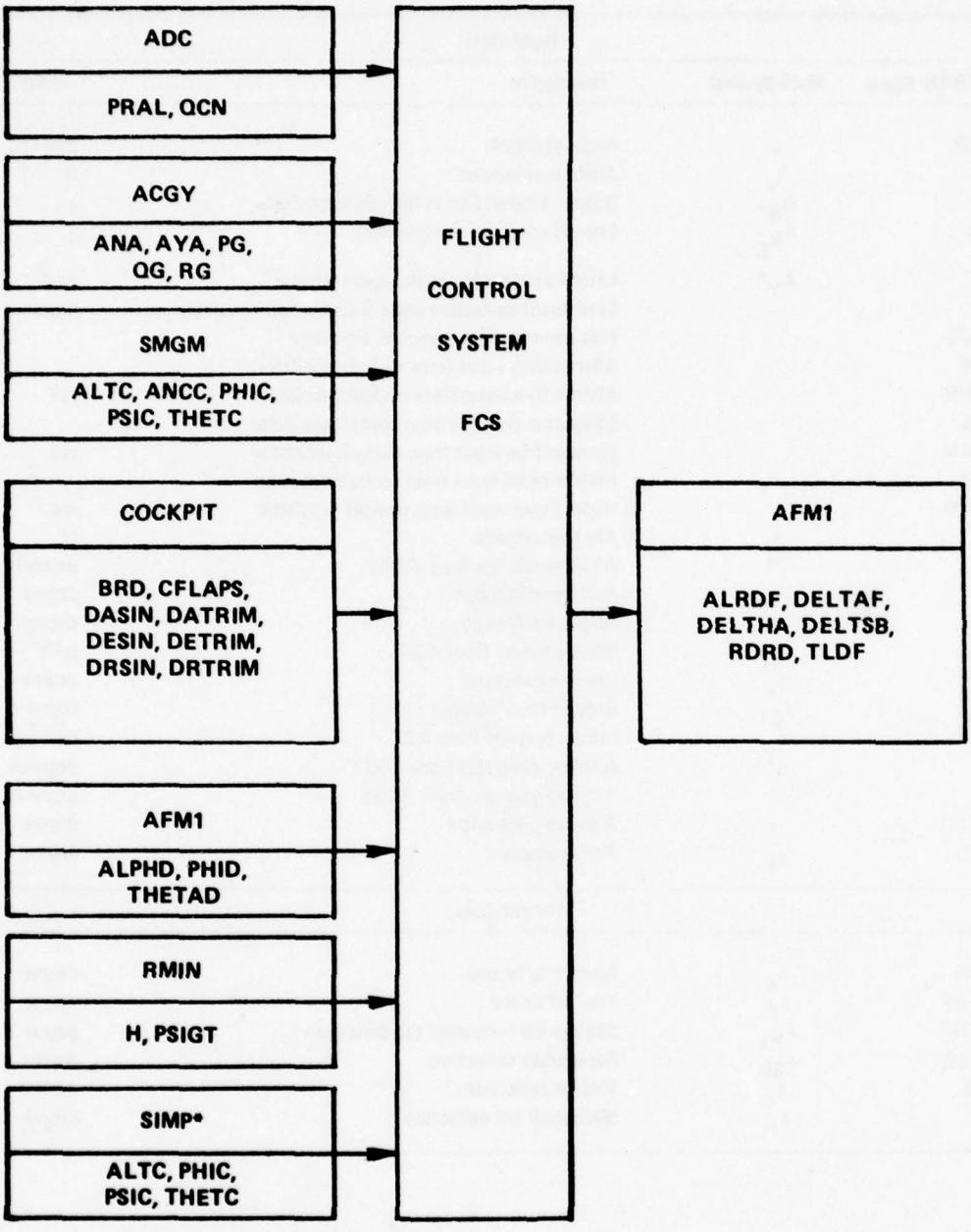
channel, while yaw rate, roll rate, and lateral acceleration are used in the lateral-directional channel.

The command following autopilot accepts guidance signals from the SMGM to fly the airplane when the cockpit is not utilized. The autopilot commands normal acceleration and roll rate through the control augmentation system to null the guidance signals. The value of the logic keys KEYLAP and KEYDAP determine if the autopilot is utilized and, if so, which command signals from the SMGM to implement. The logic definition is shown below.

KEYLAP (Key Longitudinal Autopilot)
=1 No longitudinal autopilot
=2 Altitude command
=3 Pitch attitude command
=4 Incremental pitch attitude command

KEYDAP (Key Directional Autopilot)
=1 No directional autopilot
=2 Bank angle command
=3 Heading command
=4 Incremental heading command

Figure 2.6.2.2-1 is an Input/Output Block Diagram of FCS with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.2-1.



*When mission is directed by SMGM.

Figure 2.6.2.2-1. FCS input/output block diagram.

TABLE 2.6.2.2-1. FCS INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
ALPHD	α	Angle of attack	degree
ALTC	h_c	Altitude command	ft
ANA	A_{N^*}	Normal acceleration at the accelerometer	g
ANCC	$A_{N_C'}$	Normal acceleration command	g
AYA	$A_Y{}^*$	Lateral acceleration at the accelerometer	ft/s^2
BRD	–	Speedbrake deflection angle from cockpit simulator	degree
CFLAPS		Flap command from cockpit simulator	
DASIN		Aileron stick input from cockpit simulator	
DATRIM		Aileron trim input from cockpit simulator	rad
DESIN		Elevator stick input from cockpit simulator	
DETRIM		Elevator trim input from cockpit simulator	rad
DRSIN		Rudder pedal input from cockpit simulator	
DRTRIM		Rudder trim input from cockpit simulator	rad
H	h	Airplane altitude	ft
PG	P^*	Airplane roll rate from ACGY	degree/s
PHIC	ϕ_c	Roll command signal	degree
PHID	ϕ	Airplane roll angle	degree
PRAL	P_{Si}	Static pressure from ADC	lb/ft^2
PSIC	ψ_c	Heading command	degree
PSIGT	ψ_{GT}	Ground track heading	degree
QCN	q_c	Impact pressure from ADC	lb/ft^2
QG	q^*	Airplane pitch rate from ACGY	degree/s
RG	r^*	Airplane yaw rate from ACGY	degree/s
THETAD	ϕ	Airplane pitch angle	degree
THETC	ϕ_c	Pitch command	degree
		Output Data	
ALRDF	δ_a	Aileron deflection	degree
DELTAF	δ_F	Flap deflection	degree
DELTHA	δ_{Ha}	Differential horizontal tail deflection	degree
DELTSB	δ_{SB}	Speedbrake deflection	degree
RDRD	δ_r	Rudder deflection	degree
TLDF	δ_H	Horizontal tail deflection	degree

2.6.2.3 Air Data Computer Model (ADC)

The ADC subprogram is the mathematical model of the air data computer and important air data sensor dynamics. The ability to degrade their performance of the model through the use of the NERR model on sensor inputs is included.

The purpose of the Air Data Computer model mathematical operation is to provide air data sensor information to an actual air data computer (hardware) and/or use the same information in simulating air data computer functions. The major functions in the ADC subprogram area are to:

1. Provide the air data sensor inputs:
   Air Data Computer Model,
   Static pressure,
   Total pressure, and
   Indicated temperature; and

2. Calculate the air data outputs:
   Pressure altitude,
   Pressure altitude rate,
   Mach number,
   True airspeed,
   Free air temperature, and
   Indicated airspeed.

The air data sensors provide the signals necessary for the air data computer to calculate its output quantities. Idealized sensor signals are either received from other models or calculated from input quantities.

Static pressure and free air temperature come from the Atmosphere model as standard day table look-up values. Angle of attack and sideslip angles are calculated in the Airframe model from airplane velocity components. The other air data input, impact pressure, is calculated in the sensor model as a function of the Mach number obtained from the airframe model. After these signals are shaped in the sensor module they enter the Air Data Computer module as sensor output signals. These signals are driven through a lag transfer function to simulate system lags. Sensor noise and bias errors may be incorporated by setting IRNADC = 1 and defining the rms noise and bias levels.

The air data computer calculates the desired air data output parameters from the sensor signals. These computations are performed in flight by an analog or digital computer utilizing equations similar to those implemented in this simulation.

Figure 2.6.2.3-1 is an Input/Output Block Diagram of ADC with the source of each

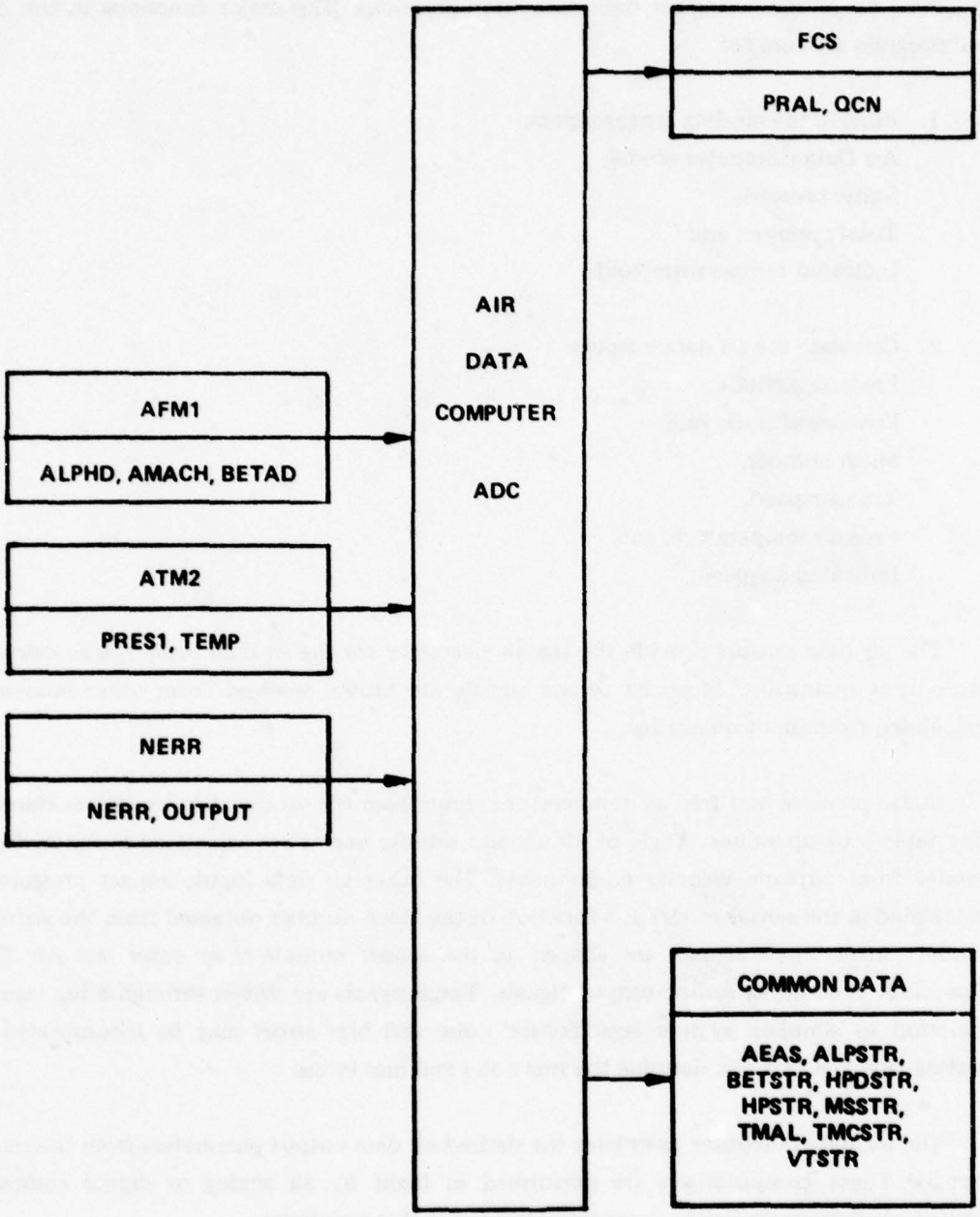


Figure 2.6.2.3-1. ADC input/output block diagram.

input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.3-1 below.

2.6.2.4 Accelerometers/Gyros Model (ACGY)

ACGY simulates the linear accelerometers and gyros by corrupting "true" acceleration, roll, pitch and yaw. It considers sensor location on the airplane, and significant sensor characteristics (deadbands, limits, noise, and bias). These characteristics may be specified or bypassed by setting the logic key KEYAG to one or zero, respectively.

The Accelerometers and Gyros sensor simulation consist of three submodules. One submodule accounts for the effects of locating the accelerometers at other than the center of gravity of the vehicle. This module is utilized regardless of the value of KEYAG. The other two submodules contain the significant sensor characteristics, which may be specified or bypassed. Random noise and bias errors are included in the model by specifying the appropriate logic keys and the rms noise and bias levels. Noise and bias are specified for each individual sensor.

TABLE 2.6.2.3-1. ADC INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
ALPHD	α	Angle of attack	degree
AMACH	M	Mach no.	
BETAD	β	Sideslip angle	degree
PRES1	P_s	Barometric pressure	lb/ft^2
TEMP	T_m	Air temperature	$^\circ R$
		Output Data	
AEAS	Vi^*	ADC indicated air speed	kt
ALPSTR	α^*	ADC angle of attack	degree
BETSTR	β^*	ADC sideslip angle	degree
HPDSTR	$\dot{H}p^*$	ADC pressure altitude rate	ft/s
HPSTR	Hp^*	ADC pressure altitude	ft
MSTR	M^*	ADC Mach no.	
PRAL	P_{si}	ADC static pressure	lb/ft^2
QCN	qc	ADC impact pressure	lb/ft^2
TMAL	Tm_{R^*}	ADC free air temperature (Rankine)	$^\circ R$
TMCSTR	T_{mc^*}	ADC free air temperature (Centigrade)	$^\circ C$
VTSTR	V_{T^*}	ADC true air speed	kt

ACGY first determines "true" (i.e., uncorrupted) values of the normal acceleration ANI and the lateral acceleration AYI at the sensor location, using the sensor displacements DXAN and DYAN from the center of gravity of the plane, and the true values of normal and lateral acceleration (at the center of gravity) input from the Airframe Model. It also scales the input true values of the angular roll rate p, pitch rate q, and yaw rate r from radians/second to degrees/second. ACGY then corrupts ANI, AYI, and pitch, roll, and yaw rates (QG, PG, and RG) by introducing errors due to deadbands, limits, noise, and bias. The subroutine contains options to bypass the error computations. The options are selected by setting the flags KEYAG and IRNAEG, as follows:

1. If KAYAG=1, errors based on the sensor deadbands and limits are computed.
2. If KEYAG=0, deadbands are not computed.
3. If IRNAEG=1, errors based on the sensor noise and bias are computed.
4. If IRNAEG=0, these errors are not computed.

The accelerations (either normal or lateral) as corrupted by the sensor's deadband are determined by the function DBND. Using this function, the acceleration is set to zero if the input true value of acceleration is within the deadband. If the input true value is outside the deadband, the output value is found by subtracting the deadband from the input true value.

Figure 2.6.2.4-1 is an Input/Output Block Diagram of ACGY with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.4-1 below.

2.6.2.5 Simulated Pilot Model (SIMP)

This model provides simulated pilot steering dynamics to modify the steering commands to the flight control system when the mission is directed by the synthetic mission generator (SMGM). The Simulated Pilot model for the simulation program is represented as a single module. Autopilot guidance command signals are modified in the pilot model to simulate the human pilot steering dynamics. The model includes a transport lag, a neuromuscular lag, a lead-lag function and a gain parameter. All inputs to the simulated pilot originate in the Synthetic Mission Generator model. Subroutine SIMP is called by subroutine DIR when SIMP is required. SIMP communicates with SMGM, CALP and FCS through common variables. Outputs from SIMP are the command signals which are put into the common data file and subsequently are available to the flight control system.

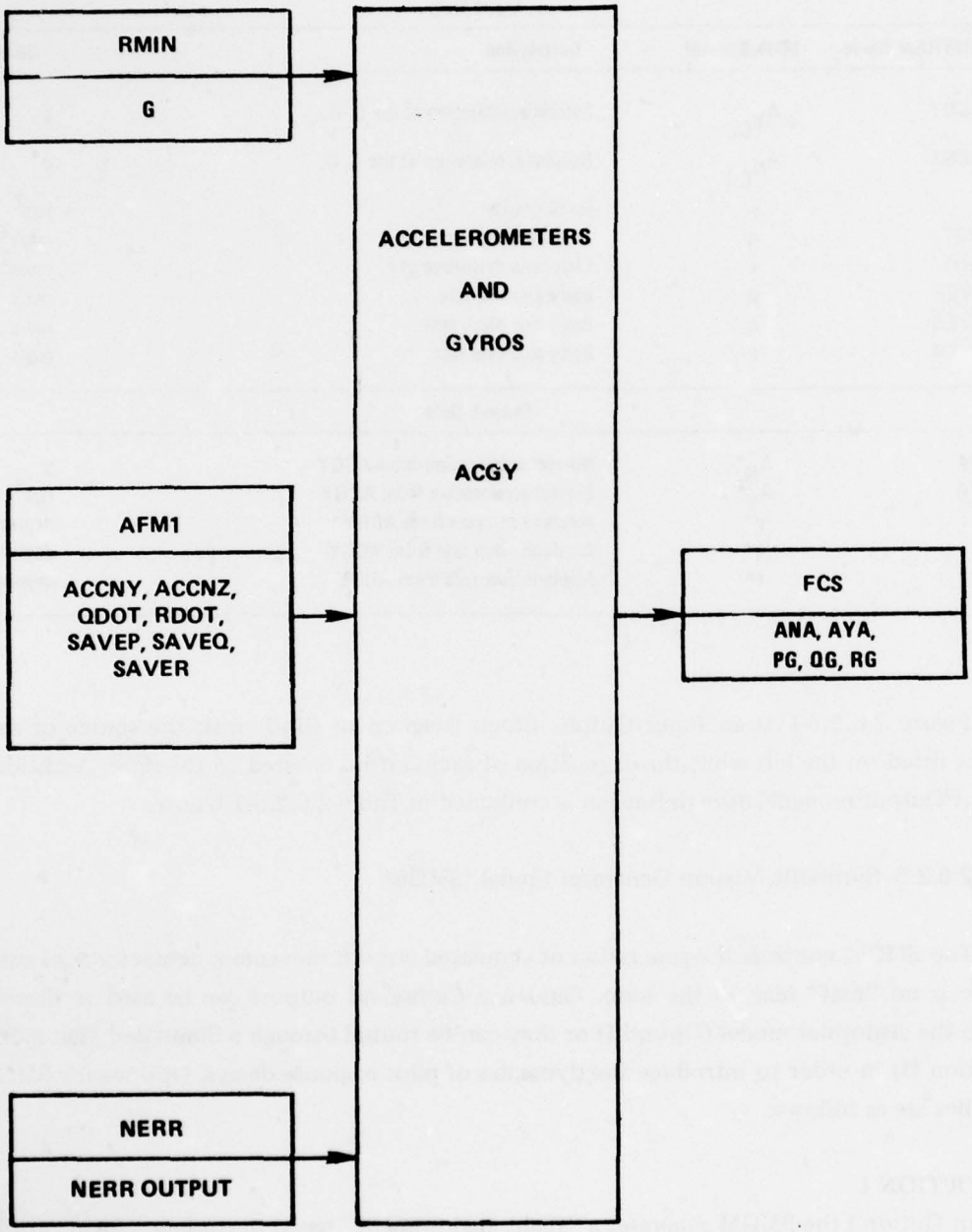


Figure 2.6.2.4-1. ACGY input/output block diagram.

TABLE 2.6.2.4-1. ACGY INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
ACCNY	$A_{Y_{CG}}$	Lateral acceleration at the C. G.	g
ACCNZ	$A_{N_{CG}}$	Normal acceleration at the C. G.	g
G	g	Earth gravity	ft/s^2
QDOT	$\dot{q}$	First time derivative of q	rad/s^2
RDOT	$\dot{r}$	First time derivative of r	rad/s^2
SAVEP	p	Body axis roll rate	rad/s
SAVEQ	q	Body axis pitch rate	rad/s
SAVER	r	Body axis yaw rate	rad/s
		Output Data	
ANA	A_N^*	Normal acceleration from ACGY	g
AYA	A_Y^*	Lateral acceleration from ACGY	ft/s^2
PG	p^*	Airplane roll rate from ACGY	degree/s
QG	q^*	Airplane pitch rate from ACGY	degree/s
RG	r^*	Airplane yaw rate from ACGY	degree/s

Figure 2.6.2.5-1 is an Input/Output Block Diagram of SIMP with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.5-1 below.

2.6.2.6 Synthetic Mission Generator Model (SMGM)

The SMGM controls the generation of simulated aircraft movement characteristics when there is no "real" man in the loop. Guidance Command outputs can be used to directly drive the Autopilot model (Option I) or they can be routed through a Simulated Man model (Option II) in order to introduce the dynamics of pilot response delays. Options for SMGM profiles are as follows:

1. OPTION I

   In Option I the SMGM generates a "flight environment" based on an input flight profile. In this option, the airframe model is used to supply the following inputs to the Reference Model for Inertial Navigation (RMIN):

   a. airplane pitch angle,

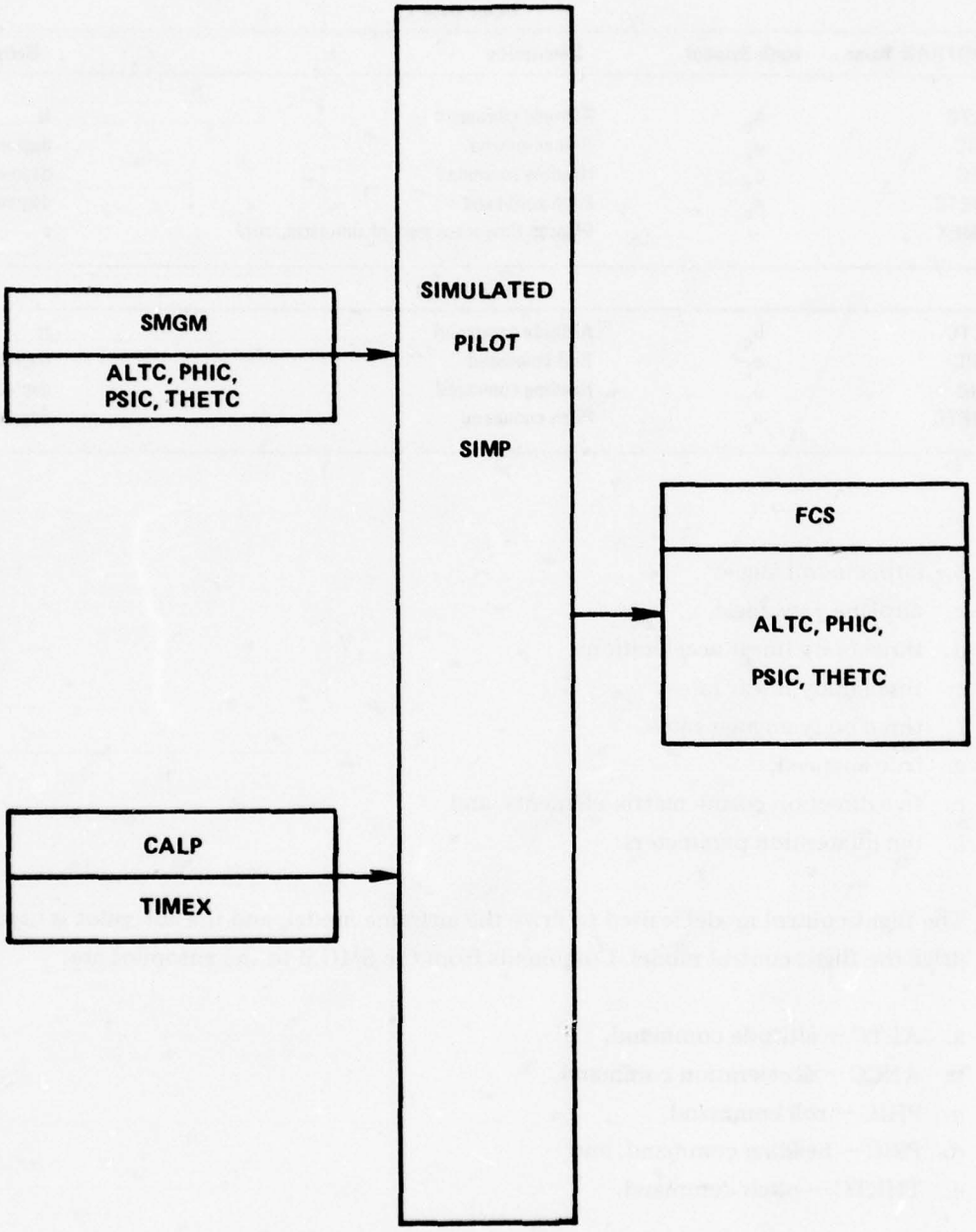


Figure 2.6.2.5-1. SIMP input/output block diagram.

TABLE 2.6.2.5-1. SIMP INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
ALTC	h_c	Altitude command	ft
PHIC	Φ_c	Roll command	degree
PSIC	ψ_c	Heading command	degree
THETC	θ_c	Pitch command	degree
TIMEX	–	Elapsed time since start of simulation run	s
		Output Data	
ALTC	h_c	Altitude command	ft
PHIC	Φ_c	Roll command	degree
PSIC	ψ_c	Heading command	degree
THETC	θ_c	Pitch command	degree

b. airplane roll angle,
c. airplane yaw angle,
d. three body linear accelerations,
e. three body linear rates,
f. three body angular rates,
g. true airspeed,
h. five direction cosine matrix elements, and
i. ten quaternion parameters.

The flight control model is used to drive the airframe model, and the autopilot is used to drive the flight control model. Commands from the SMGM to the autopilot are:

a. ALTC – altitude command,
b. ANCC – acceleration command,
c. PHIC – roll command,
d. PSIC – heading command, and
e. THETC – pitch command.

In addition to the above commands, the velocity command, VELC, from SMGM is channeled directly to the airframe where VELC is the control input for the autothrottle.

2. OPTION II

Option II is the same as Option I except that the response delays of a simulated pilot shall be applied to commands prior to feeding the commands to the autopilot.

Figure 2.6.2.6-1 is an Input/Output Block Diagram of SMGM with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.6-1 below.

2.6.2.7 Target Simulation (TGT)

The Target model is a service model that functions with the Relative Geometry model, Weather model and the sensor models. The function of the Target model is to permit simulated target positions and signatures to be sent to the Target Detection Sensor models (i.e., radar and E/O). The target characteristics simulated include target and background radar cross sections and background radiances. The model functions include derivations of target positions from movement profiles for the simulation of moving targets and computation of radar background cross sections. Specifically, it performs the following logical functions:

1. It provides the location of each non-moving target to the Relative Geometry model.
2. It computes and provides the location of each moving target to the Relative Geometry model.
3. It computes the radar ground return cross section to be supplied to the Radar model.
4. It provides the logic to control the comparison of target location with sensor coverage in order to minimize the number of times the in coverage test must be performed.

The location of each non-moving target is TGT model input data and is stored in the data common where it is available for use by the Relative Geometry model. The non-moving target location data is provided in terms of latitude, longitude, and altitude above sea level. The target model shall generate the current location of the moving targets in accordance with a moving target profile which is stored as initialization data. The initial position of the target is specified in terms of latitude and longitude. The target profile shall commence when the initial point is within some maximum range RMAX2. The current location of the moving target is then computed and passed to relative geometry in terms of latitude, longitude, and altitude.

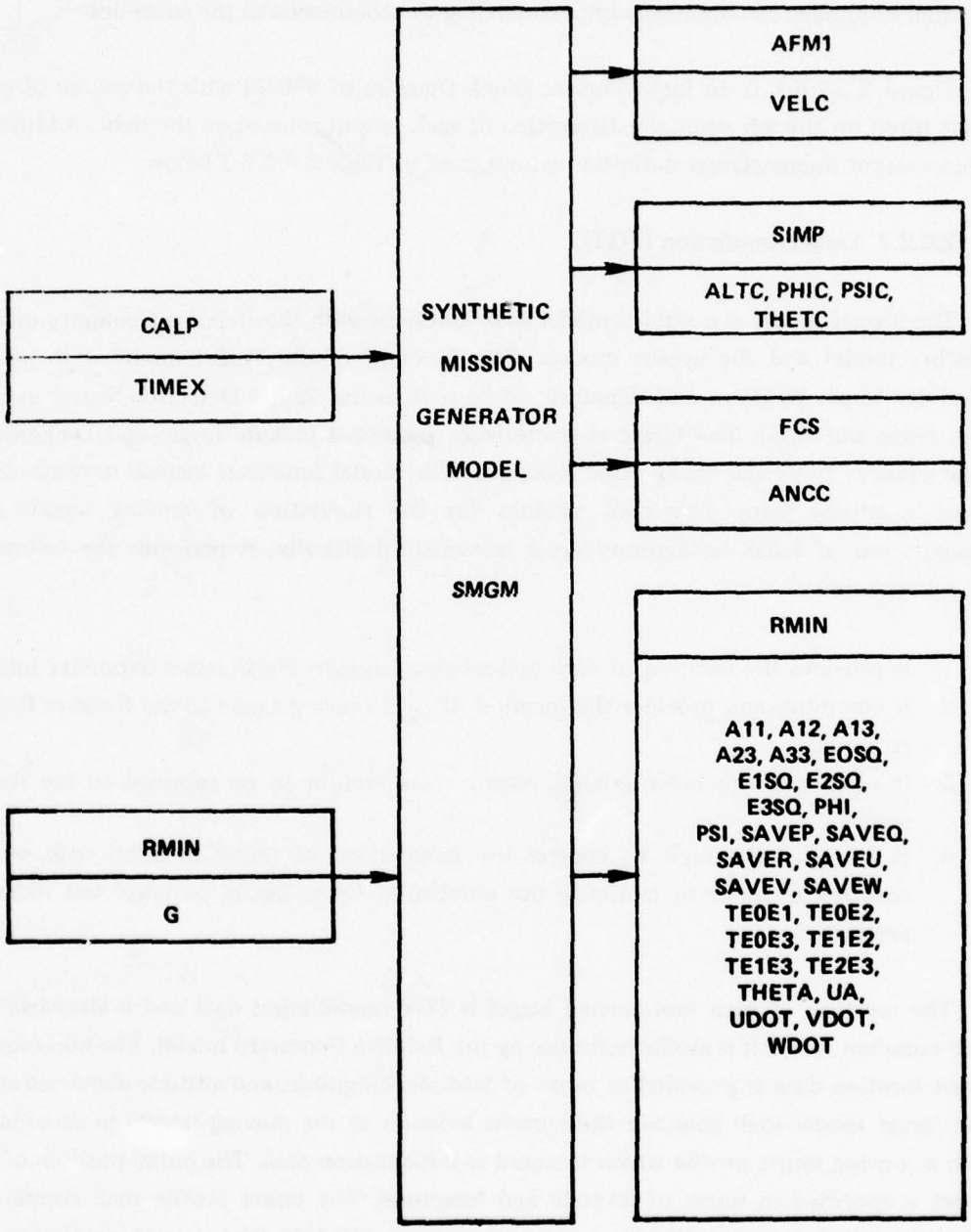


Figure 2.6.2.6-1. SMGM input/output block diagram.

TABLE 2.6.2.6-1. SMGM INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
G	g	Earth gravity	ft/s^2
TIMEX	t	Elapsed time since start of simulation run	s
		Output Data	
A11	a_{11}	Direction cosine matrix element	
A12	a_{12}	Direction cosine matrix element	
A13	a_{13}	Direction cosine matrix element	
A23	a_{23}	Direction cosine matrix element	
A33	a_{33}	Direction cosine matrix element	
ALTC	h_c	Altitude command	ft
ANCC	$A_{N_{c'}}$	Normal acceleration command	g
EOSQ	$(E_0)^2$	E_0 squared	
E1SQ	$(E_1)^2$	E_1 squared	
E2SQ	$(E_2)^2$	E_2 squared	
E3SQ	$(E_3)^2$	E_3 squared	
PHI	ϕ	Airplane roll angle	rad
PHIC	ϕ_c	Roll command signal	degree
PSI	ψ	Airplane yaw angle	rad
PSIC	ψ_c	Heading command	degree
SAVEP	p	Body axis roll rate	rad/s
SAVEQ	q	Body axis pitch rate	rad/s
SAVER	r	Body axis yaw rate	rad/s
SAVEU	u	X axis airplane velocity WRT earth	ft/s
SAVEV	v	Y axis airplane velocity WRT earth	ft/s
SAVEW	w	Z axis airplane velocity WRT earth	ft/s
TE0E1	–	Two x E0 x E1	
TE0E2	–	Two x E0 x E2	
TE0E3	–	Two x E0 x E3	
TE1E2	–	Two x E1 x E2	
TE1E3	–	Two x E0 x E3	
TE2E3	–	Two x E2 x E3	
THETA	θ	Airplane pitch angle	rad
THETC	θ_c	Pitch command	degree
UA	UA	True air speed	ft/s
UDOT	$\dot{u}$	X axis airplane acceleration WRT earth	ft/s^2
VDOT	$\dot{v}$	Y axis airplane acceleration WRT earth	ft/s^2
VELC	Vc	Airplane velocity command	ft/s^2
WDOT	$\dot{w}$	Z axis airplane acceleration WRT earth	ft/s^2

Figure 2.6.2.7-1 is an Input/Output Block Diagram of TGT with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.7-1 below.

2.6.2.8 Attack Radar (ARS)

The Attack Radar model is a model that functions with the service and reference models to produce a simulation of a generic attack radar. The model simulates the following modes and functions of the attack radar:

1. Air-to-Ground Search,
2. Air-to-Ground Ranging,
3. Air-to-Ground Level Bombing/Fixtaking, and
4. Air-to-Air.

In the air-to-ground search mode, ARS provides the following functions:

1. Drives the antenna in a simulated search scan,
2. Determines signal level of any targets within the radar beam,
3. Computes radar range to target, and
4. Computes noise output signal, weather signal, and signal + background + noise.

In the air-to-ground ranging mode, ARS provides the following functions:

1. Generates antenna azimuth and elevation pointing signals,
2. Drives the antenna to the simulated azimuth and elevation angles,
3. Determines signal level returned from the target,
4. Computes radar range to target, and
5. Computes noise output signal, weather signal, and signal + background + noise.

In the air-to-ground level bombing/fixtaking mode, ARS provides the following functions:

1. Accepts cursor positioning commands from the cockpit,
2. Drives the antenna in a simulated search scan,
3. Determines signal level of any targets within radar beam,
4. Computes radar range to target, and
5. Computes noise output signal, weather signal, and signal + background + noise.

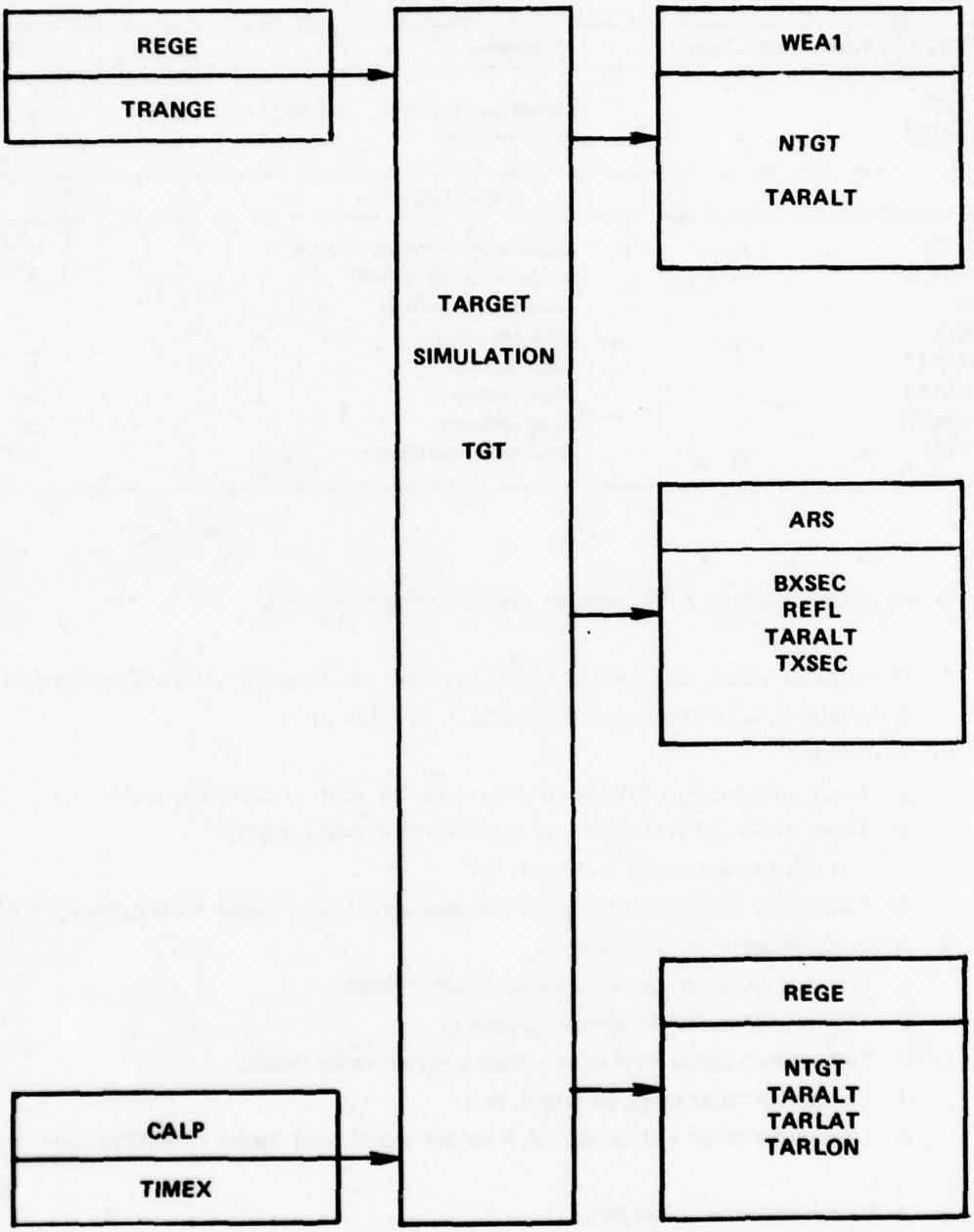


Figure 2.6.2.7-1. TGT input/output block diagram.

TABLE 2.6.2.7-1. TGT INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
TIMEX	t	Elapsed time since start of simulation run	s
TRANGE	–	Range to target	ft
		Output Data	
BXSEC	σ_B	Radar background cross section	ft^2
NTGT	–	Number of targets defined	
ON	–	Target in-coverage flag	
REFL	γ_R	Radar rain reflection	
TARALT	–	Target altitude	ft
TARLAT	–	Target latitude	rad
TARLON	–	Target longitude	rad
TXSEC	σ_T	Target radar cross section	ft^2

In the air-to-air mode, ARS provides the following functions:

1. Determines which air-to-air submode is active (A/A search, A/A acquisition/manual designate, A/A acquisition/autodesignate, A/A track);
2. A/A search
   a. Generates antenna scan positions (single-bar scan or multi-bar scan),
   b. Determines signal level of any targets within radar beam,
   c. Computes radar range to target, and
   d. Computes noise output signal, weather signal, and signal + background + noise;
3. A/A acquisition/manual designate
   a. Accepts cursor positioning signal from cockpit,
   b. Drives antenna to designated position,
   c. Determines signal level of any target within radar beam,
   d. Computes radar range to target, and
   e. Computes noise output signal, weather signal, and signal + background + noise;
4. A/A acquisition/autodesignate
   a. Places cursors upon nearest target,
   b. Drives antenna to designated position,
   c. Determines signal level of any targets within radar beam,
   d. Computes radar range to target, and

e. Computes noise output signal, weather signal, and signal + background + noise;

5. A/A track
   a. Locks antenna on predesignated target,
   b. Determines signal level of target,
   c. Computes radar range to target, and
   d. Computes noise output signal, weather signal, and signal + background + noise.

Figure 2.6.2.8-1 is an Input/Output Block Diagram of ARS with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.8-1 below.

2.6.2.9 Radar Altimeter (RALT)

The Radar Altimeter model is a sensor model that functions with the service and reference models to produce a simulation of a generic radar altimeter. Provision is made to vary the radar altimeter antenna parameters and the roll and pitch limits through which the altimeter will function without large error. Provision is made to simulate both random and bias errors .

The inputs to the radar altimeter consist of true values of airplane pitch and roll angles, altitude above sea level, elapsed time, and altimeter on/off signal. RALT begins by determining whether aircraft attitude enables an accurate measurement by a radar altimeter. If the aircraft is to receive an altitude measure, first the true value of altitude above the earth plane is determined; second, this value is corrupted as in a hardware radar altimeter; and third, an "altitude-good" is returned to the system. Output data from the radar altimeter model consists of the radar altitude above terrain and the output-good signal. All data is transferred via common blocks. Figure 2.6.2.9-1 is an Input/Output Block Diagram of RALT with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.9-1 below.

2.6.2.10 Random Noise and Error Generation (NERR)

The function of the Normal Error Generator is to provide random and bias error variates to perturbate simulation signals in other models. The model employs a pseudo-random number generator to produce the normal variates generated and stored in an off-line (non-real-time) initialization program. The real-time program calls the normal variates from

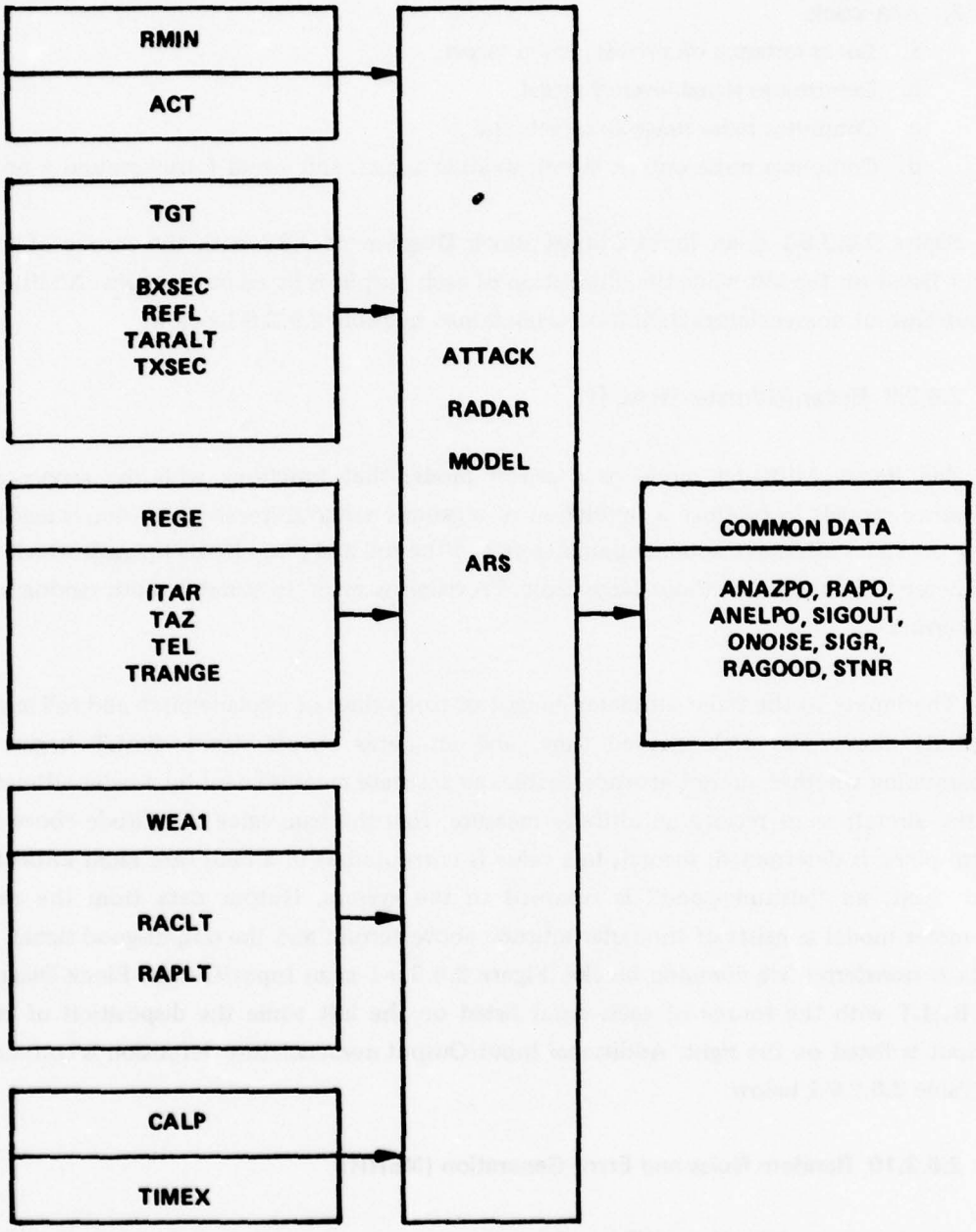


Figure 2.6.2.8-1. ARS input/output block diagram.

TABLE 2.6.2.8-1. ARS INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
ALT	h	Airplane altitude	ft
BXSEC	σ_B	Radar background cross section	ft^2
ITAR	–	Selected target	
RACLT	–	Length of LOS segment to target in layer 2	ft
RAPRT	–	Length of LOS segment to target in layer 3	ft
REFL	γ_R	Radar rain reflection coefficient	
TARALT	–	Target altitude	ft
TAZ	–	Target body azimuth angle	rad
TEL	–	Target body elevation angle	rad
TIMEX	t	Elapsed time since start of simulation run	s
TRANGE	–	Range to target	ft
TXSEC	σ_T	Target radar cross section	ft^2
		Output Data	
ANAZPO	–	Azimuth position of antenna	rad
ANELPO	–	Elevation position of	rad
ONOISE	–	Output noise voltage	V
RAGOOD	–	Radar good flag	
RAPO	–	Output range to target	ft
SIGOUT	–	Output signal voltage	V
SIGR	–	Weather signal output voltage	V
STNR	–	Signal-to-noise ratio	

TABLE 2.6.2.9-1. RALT INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
H	h	Airplane altitude	ft
PHI	ϕ	Airplane roll angle	rad
RAHSTA	–	Radar altimeter on/off signal	
THETA	θ	Airplane pitch angle	rad
TIMEX	t	Elapsed time since start of simulation run	s
		Output Data	
RAH	H	Radar altimeter output altitude	ft
IALTGO	–	Radar altimeter output-good signal	

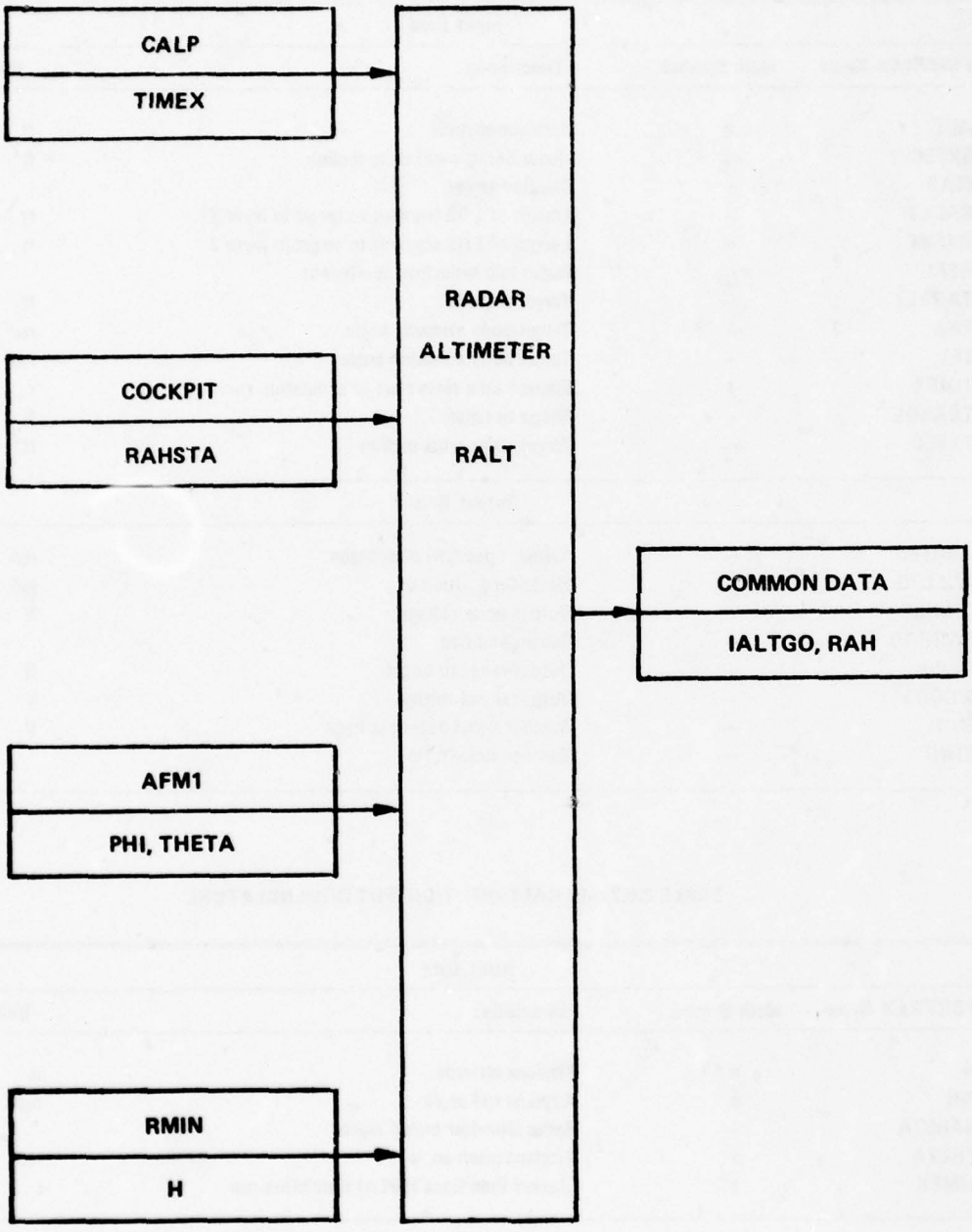


Figure 2.6.2.9-1. RALT input/output block diagram.

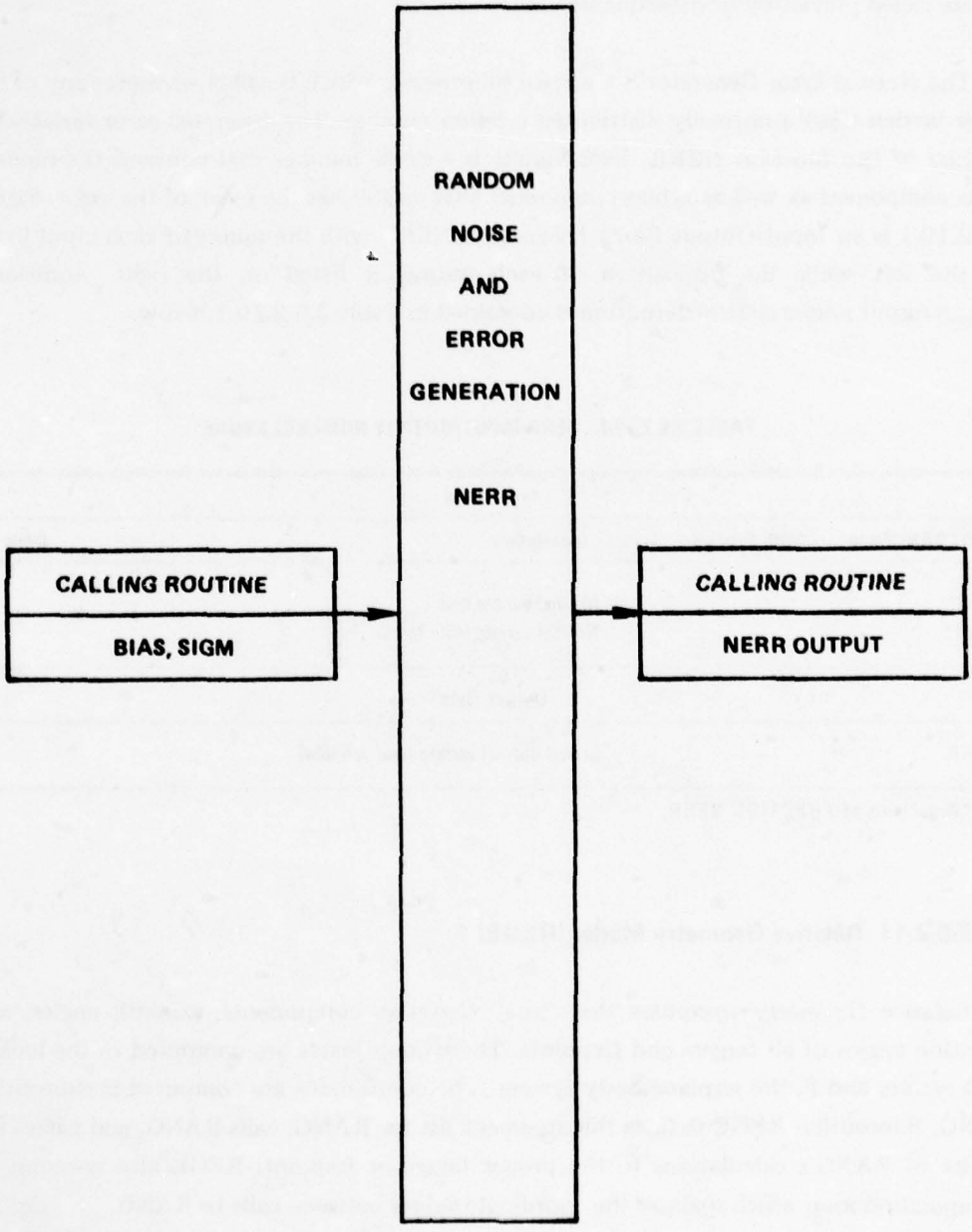


Figure 2.6.2.10-1. NERR input/output block diagram.

storage, performs a scaling operation, and adds the bias error to the called error variate. Each variate called by the normal error program is essentially independent of any other variate called previously or subsequently.

The Normal Error Generator is a service subprogram which is called whenever any of the other models need a normally distributed random number. The generated error variates are outputs of the function NERR. Each variate is a single number that contains the random noise component as well as a bias component that establishes the mean of the error. Figure 2.6.2.10-1 is an Input/Output Block Diagram of NERR with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.10-1 below.

TABLE 2.6.2.10-1. NERR INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
BIAS*	β	Normal variate bias	
SIGM*	μ	Normal variate scale factor	
		Output Data	
NERR	–	Scaled normal variate (real variable)	

* Arguments of FUNCTION NERR.

2.6.2.11 Relative Geometry Model (REGE)

Relative Geometry computes the "true" Cartesian components, azimuth angles, and elevation angles of all targets and fixpoints. These coordinates are computed in the locally level system and in the airplane body system. The coordinates are computed in subroutine RANG. Subroutine REGE defines the argument list for RANG, calls RANG, and passes the results of RANG's calculations to the proper target or fixpoint. REGE also contains an extrapolation loop which updates the coordinate values between calls to RANG.

Figure 2.6.2.11-1 is an Input/Output Block Diagram of REGE with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.11-1 below.

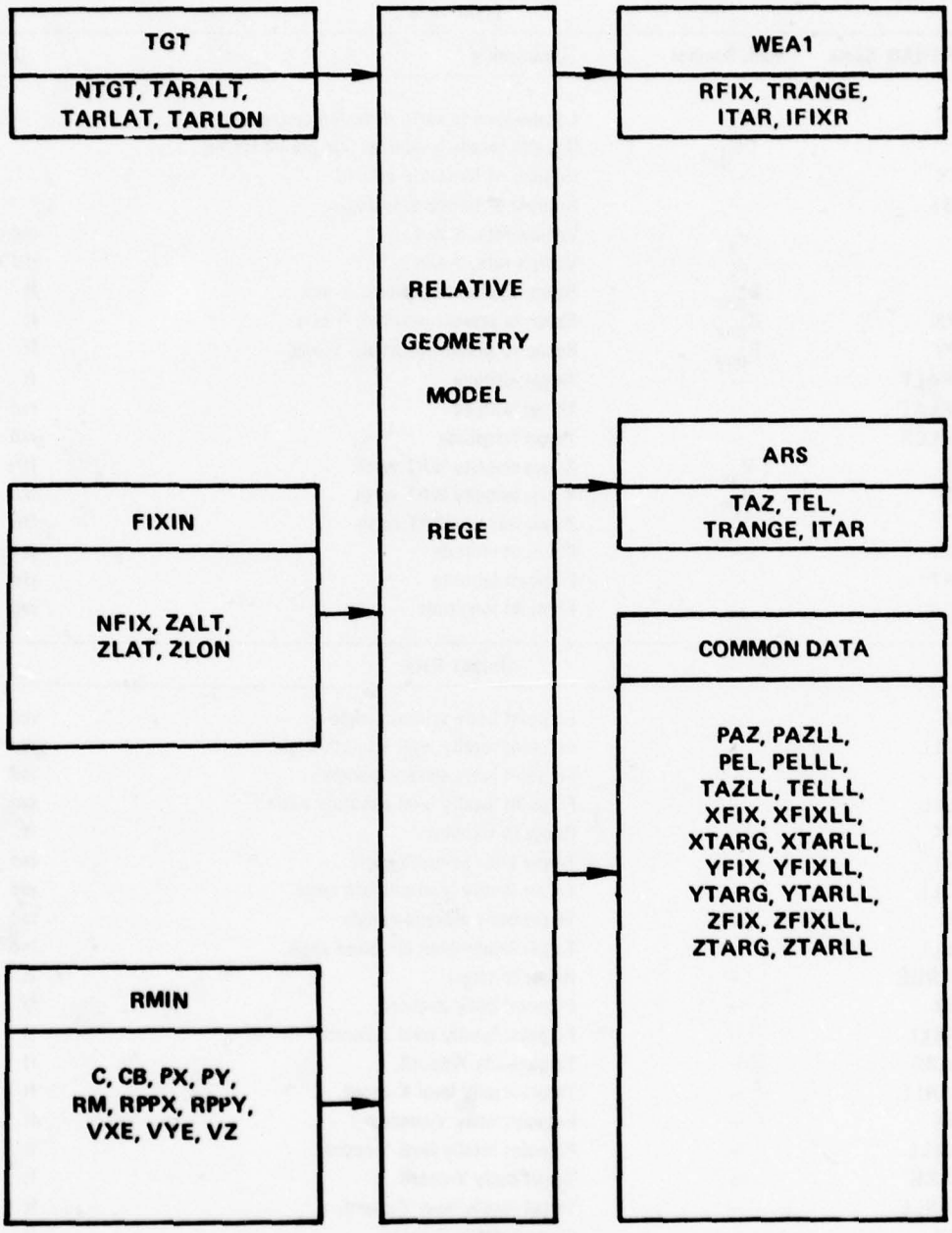


Figure 2.6.2-11-1. REGE input/output block diagram.

TABLE 2.6.2.11-1. REGE INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
C	C_{ij}	Locally level-to-earth direction cosine matrix	
CB	CB_{ij}	Body-to-locally level direction cosine matrix	
NFIX	–	Number of fixpoints defined	
NTGT	–	Number of targets defined	
PX	ρ_x	Vehicle rate, X axis	rad/s
PY	ρ_y	Vehicle rate, Y axis	rad/s
RM	R_{ppz}	Range to present position, Z axis	ft
RPPX	R_{ppx}	Range to present position, X axis	ft
RPPY	R_{ppy}	Range to present position, Y axis	ft
TARALT	–	Target altitude	ft
TARLAT	–	Target latitude	rad
TARLON	–	Target longitude	rad
VXE	V_{xe}	X axis velocity WRT earth	ft/s
VYE	V_{ye}	Y axis velocity WRT earth	ft/s
VZ	V_z	Z axis velocity WRT earth	ft/s
ZALT	–	Fixpoint altitude	ft
ZLAT	–	Fixpoint latitude	rad
ZLON	–	Fixpoint longitude	rad
		Output Data	
PAZ	–	Fixpoint body azimuth angle	rad
PAZLL	–	Fixpoint locally level azimuth angle	rad
PEL	–	Fixpoint body elevation angle	rad
PELLL	–	Fixpoint locally level elevation angle	rad
RFIX	–	Range to fixpoint	ft
TAZ	–	Target body azimuth angle	rad
TAZLL	–	Target locally level azimuth angle	rad
TEL	–	Target body elevation angle	rad
TELLL	–	Target locally level elevation angle	rad
TRANGE	–	Range to target	ft
XFIX	–	Fixpoint body X-coord.	ft
XFIXLL	–	Fixpoint locally level X-coord.	ft
XTARG	–	Target body X-coord.	ft
XTARLL	–	Target locally level X-coord.	ft
YFIX	–	Fixpoint body Y-coord.	ft
YFIXLL	–	Fixpoint locally level Y-coord.	ft
YTARG	–	Target body Y-coord.	ft
YTARLL	–	Target locally level Y-coord.	ft
ZFIX	–	Fixpoint body Z-coord.	ft
ZFIXLL	–	Fixpoint locally level Z-coord.	ft
ZTARG	–	Target body Z-coord.	ft
ZTARLL	–	Target locally level Z-coord.	ft

2.6.2.12 Weather (WEA1)

The weather model shall compute the lengths of the line-of-sight segments through each of three weather layers for all defined targets and fix points. The model assumes three layers of definable height and thickness.

The test engineer shall specify the thickness of each layer before the real-time simulation begins. The layers are concentric with the earth sphere and are uniform over the earth's surface. The conditions inside each layer (attenuation and scattering coefficients at various wavelengths, rainfall rates, etc.) are specified during set up; WEA 1 does not deal with these conditions and addresses only the geometry involved. Procedure WEA1TT is used to initialize this weather model.

Figure 2.6.2.12-1 is an Input/Output Block Diagram of WEA1 with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.12-1 below.

2.6.2.13 Atmosphere Simulation Model (ATM2)

The objective of the model ATM2 is to simulate atmospheric conditions that affect the flight of the F-16 aircraft, such as temperature, pressure, and wind.

ATM2 is designed to compute air temperature, pressure density and speed of sound as a function of altitude above sea level for use in the Airframe, Air Data Computer, and sensor models. All inputs are provided by the Reference model for INS control. Airplane altitude can vary from sea level to 80,000 feet in the atmospheric model. The temperature is calculated from temperature-altitude data represented by linear line segments and the altitude of interest that determines which segment is being operated on. The temperature is a model output and also an input to the calculation for the speed of sound.

Figure 2.6.2.13-1 is an Input/Output block diagram of ATM2 with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.13-1 below.

2.6.2.14 Reference Model for Inertial Navigation System (RMIN)

The Reference model for Inertial Navigation System Control (RMIN) provides true or reference values of inertial acceleration, velocity and position (latitude and longitude) for

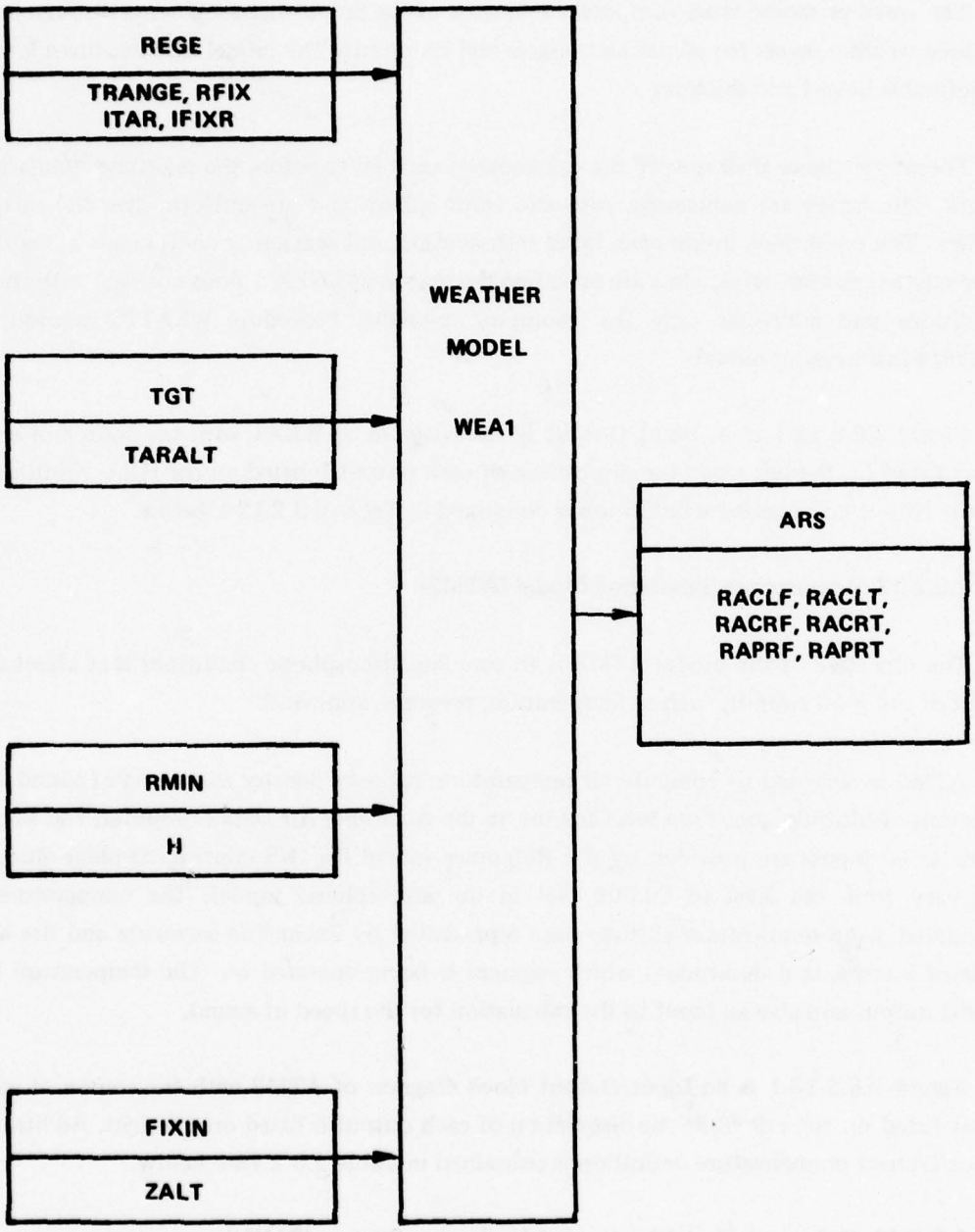


Figure 2.6.2.12-1. WEA1 input/output block diagram.

TABLE 2.6.2.12-1. WEA1 INPUT/OUTPUT NOMENCLATURE

Input Data			
FORTRAN Name	**Math Symbol**	**Description**	**Units**
H	h	Airplane altitude	ft
IFIXR	–	Selected fixpoint index	
ITAR	–	Selected target index	
RFIX	R_F	Range to fixpoint	ft
TARALT	h_T	Target altitude	ft
TRANGE	R_T	Range to target	ft
ZALT	h_F	Fixpoint altitude	ft
Output Data			
RACLF	–	Length of LOS segment to fixpoint in layer 2	ft
RACLT	–	Length of LOS segment to target in layer 2	ft
RACRF	–	Length of LOS segment to fixpoint in layer 1	ft
RACRT	–	Length of LOS segment to target in layer 1	ft
RAPRF	–	Length of LOS segment to fixpoint in layer 3	ft
RAPRT	–	Length of LOS segment to target in layer 3	ft

TABLE 2.6.2.13-1. ATM2 INPUT/OUTPUT NOMENCLATURE

Input Data			
FORTRAN Name	**Math Symbol**	**Description**	**Units**
G	g	Earth gravity	ft/s^2
H	h	Airplane altitude	ft
RM	R_{ppz}	Radius of the earth plus airplane altitude	ft
Output Data			
DENS 1	p	Air density	$slugs/ft^3$
PRES 1	P	Barometric pressure	lb/ft^2
PRESRA	P/Po	Air pressure ratio	
TEMP	T	Air temperature	$^\circ R$
TEMPRA	–	Air temperature ratio	
VS	a	Speed of sound	ft/s

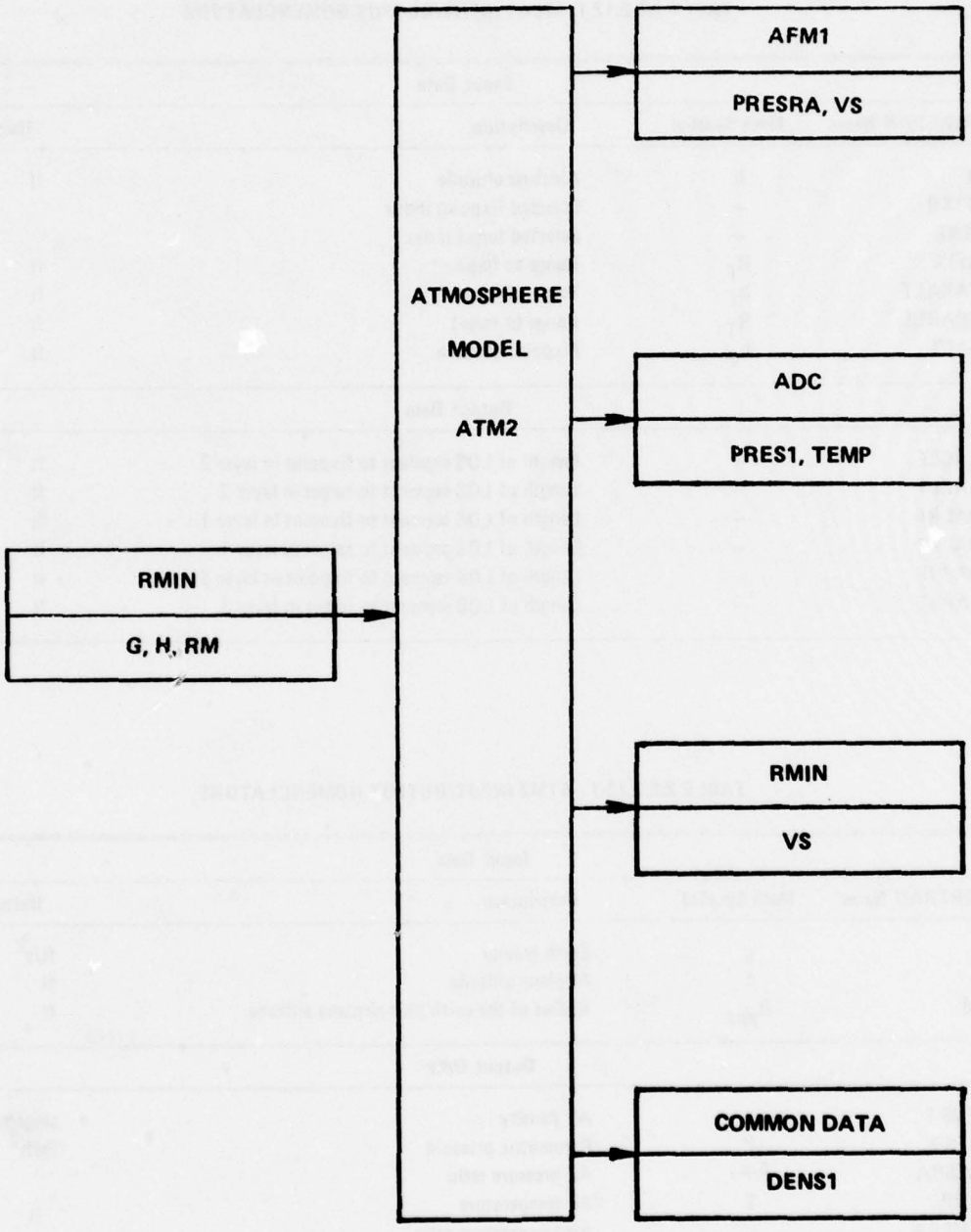


Figure 2.6.2.13-1. ATM2 input/output block diagram.

use throughout the simulation. Outputs of RMIN are accepted as those of a wander angle mechanized inertial navigator operating with no errors. These reference values are used as inputs by sensor models and are corrupted to provide indicated or sensed values. Principal inputs to RMIN are body axis linear accelerations, velocities, angular rates (referenced to a flat, nonrotating earth reference frame), direction cosines, and quaternion parameters from the Airframe model.

Initialization of RMIN is important and is accomplished by a separate routine, RMININ, before real-time execution begins. Figure 2.6.2.14-1 is an Input/Output Block Diagram of RMIN with the source of each input listed on the left while the disposition of each output is listed on the right. Additional Input/Output nomenclature definition is contained in Table 2.6.2.14-1 below.

2.6.2.15 Inertial Reference Unit (IRU)

The purpose of the IRU model is to simulate the outputs of the Inertial Reference Unit sensor components which are triads of accelerometers and gyros, with sensitive axes properly oriented, and mounted on a gimballed platform. The Inertial Reference Unit is the primary source of data for navigation of the aircraft. The accelerometers provide data from which velocity and position are determined while the gimbals provide altitude information. The gyros, together with their torquing system, aid in keeping the platform locallv level when precession occurs. Navigation information is processed from IRU sensors onboard the aircraft using a computerized navigation mechanization called an Operational Flight Program (OFP).

The IRU model simulates accelerations from the platform accelerometers and attitude angles from the platform gimbals. The IRU model takes as inputs true attitude from the airframe and locally level components of inertial acceleration from RMIN, and corrupts these true values with gyro and accelerometer (component) errors. IRU component error source parameters and platform tilts must be initialized. The outputs of the IRU model are indicated (simulated) platform accelerations and gimbal (attitude) angles for use by the Navigation OFP. The IRU model can be altered to accommodate a strapdown inertial reference unit (SIRU) during the growth phase. In a strapdown system, the gyros and accelerometers are rigidly mounted to the aircraft body axes instead of a gimballed platform. Inputs to the SIRU model would then require body axis angular rates and body axis linear accelerations instead of attitude and inertial accelerations referenced to a locally level coordinate system.

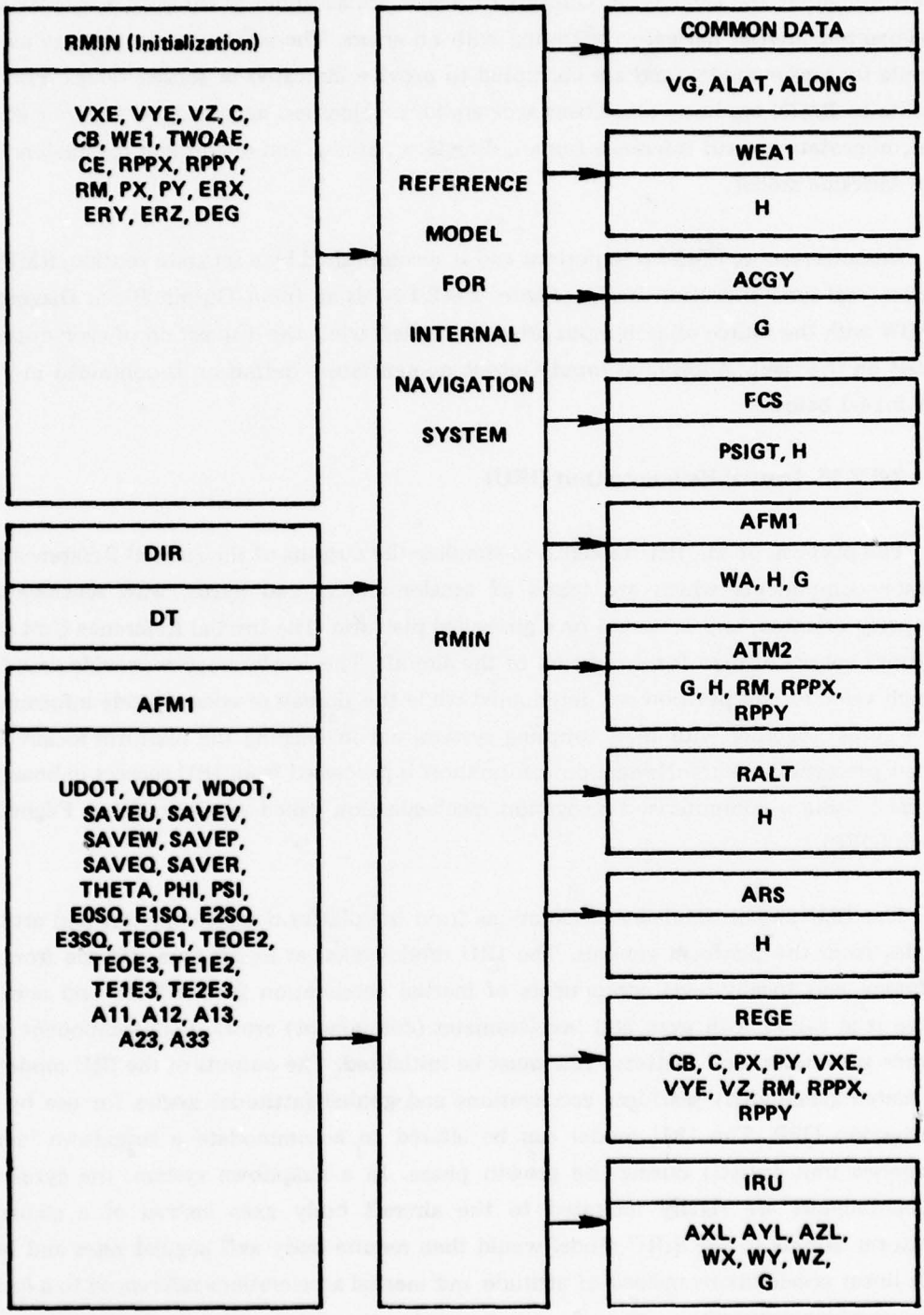


Figure 2.6.2.14-1. RMIN input/output block diagram.

TABLE 2.6.2.14-1. RMIN INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
		Input Data	
A11	a_{11}	Direction cosine matrix element	
A12	a_{12}	Direction cosine matrix element	
A13	a_{13}	Direction cosine matrix element	
A23	a_{23}	Direction cosine matrix element	
A33	a_{33}	Direction cosine matrix element	
E0SQ	$(E_0)^2$	E_0 squared	
E1SQ	$(E_1)^2$	E_1 squared	
E2SQ	$(E_2)^2$	E_2 squared	
E3SQ	$(E_3)^2$	E_3 squared	
PHI	ϕ	Airplane roll angle	rad
PSI	ψ	Airplane yaw angle	rad
SAVEP	p	Body axis roll rate	rad/s
SAVEQ	q	Body axis pitch rate	rad/s
SAVER	r	Body axis yaw rate	rad/s
SAVEU	u	X axis airplane velocity WRT earth	ft/s
SAVEV	v	Y axis airplane velocity WRT earth	ft/s
SAVEW	w	Z axis airplane velocity WRT earth	ft/s
TE0E1	–	Two x E0 x E1 (quaternion parameter)	
TE0E2	–	Two x E0 x E2 (quaternion parameter)	
TE0E3	–	Two x E0 x E3 (quaternion parameter)	
TE1E2	–	Two x E1 x E2 (quaternion parameter)	
TE1E3	–	Two x E1 x E3 (quaternion parameter)	
TE2E3	–	Two x E2 x E3 (quaternion parameter)	
THETA	θ	Airplane pitch angle	rad
UDOT	$\dot{u}$	X axis airplane acceleration with respect to earth	ft/s^2
VDOT	$\dot{v}$	Y axis airplane acceleration with respect to earth	ft/s^2
WDOT	$\dot{w}$	Z axis airplane acceleration with respect to earth	ft/s^2
		Output Data	
ALAT	γ	Present position latitude	degree
ALONG	λ	Present position longitude	degree
AXL	A_{XL}	X locally level acceleration	ft/s^2
AYL	A_{YL}	Y locally level acceleration	ft/s^2
AZL	A_{ZL}	Z locally level acceleration	ft/s^2
C	C_{ij}	Locally level-to-earth direction cosine matrix	
CB	CB_{ij}	Body-to-locally level direction cosine matrix	
G	g	Earth gravity	ft/s^2
H	h	Airplane altitude	ft
PSIDR	ψ_D	Drift angle	degree
PSIGT	ψ_G	Ground track angle	degree
PX	p_x	Vehicle rate, X axis	rad/s
PY	p_y	Vehicle rate, Y axis	rad/s
RM	R_{ppz}	Range to present position Z axis	ft
RPPX	R_{ppx}	Range to present position X axis	ft

TABLE 2.6.2.14-1. RMIN INPUT/OUTPUT NOMENCLATURE (con.)

Output Data (con.)			
FORTRAN Name	**Math Symbol**	**Description**	**Units**
RPPY	R_{ppy}	Range to present position Y axis	ft
VG	V_g	Ground speed	ft/s
VXE	V_{xe}	X axis velocity with respect to earth	ft/s
VYE	V_{ye}	Y axis velocity with respect to earth	ft/s
VZ	V_z	Z axis velocity with respect to earth	ft/s
WA	–	Wander angle	degree
WX	ω_x	X axis angular rate	rad/s
WY	ω_y	Y axis angular rate	rad/s
WZ	ω_z	Z axis angular rate	rad/s

IRU component error source models are statistical in nature, with the range of inputs being large enough to model a variety of IRUs. All modeled error sources accept outputs from a random noise generator subroutine which produces normally distributed noise that is dependent on a user-specified mean and standard deviations. All modeled IRU error sources are normally included in each iteration and run; however, the user can delete any error source variable.

Figure 2.6.2.15-1 attempts to show the interaction of this model with other elements of the real-time model set. The source model of each input is listed on the left while the disposition model is listed on the right. Additional nomenclature definition is contained in Table 2.6.2.15-1 below.

2.6.2.16 Flux Valve Model (FLUX)

The Flux Valve sensor (one of the navigation models) provides a source of magnetic heading; that is, if the magnetic variation corresponding to the present aircraft position relative to the earth is known, true north can be approximately determined by using a flux valve output. A typical flux valve output is a noisy analog signal, which is smoothed by using the AFRS directional gyroscope.

The flux valve model, FLUX, simulates a source of magnetic heading which can be used for navigation. As shown in the Input/Output Block Diagram of Figure 2.6.2.16-1, FLUX receives inputs of true heading from AFM1, magnetic variation, and flux valve noise standard deviation from the user initialization to produce flux valve magnetic heading,

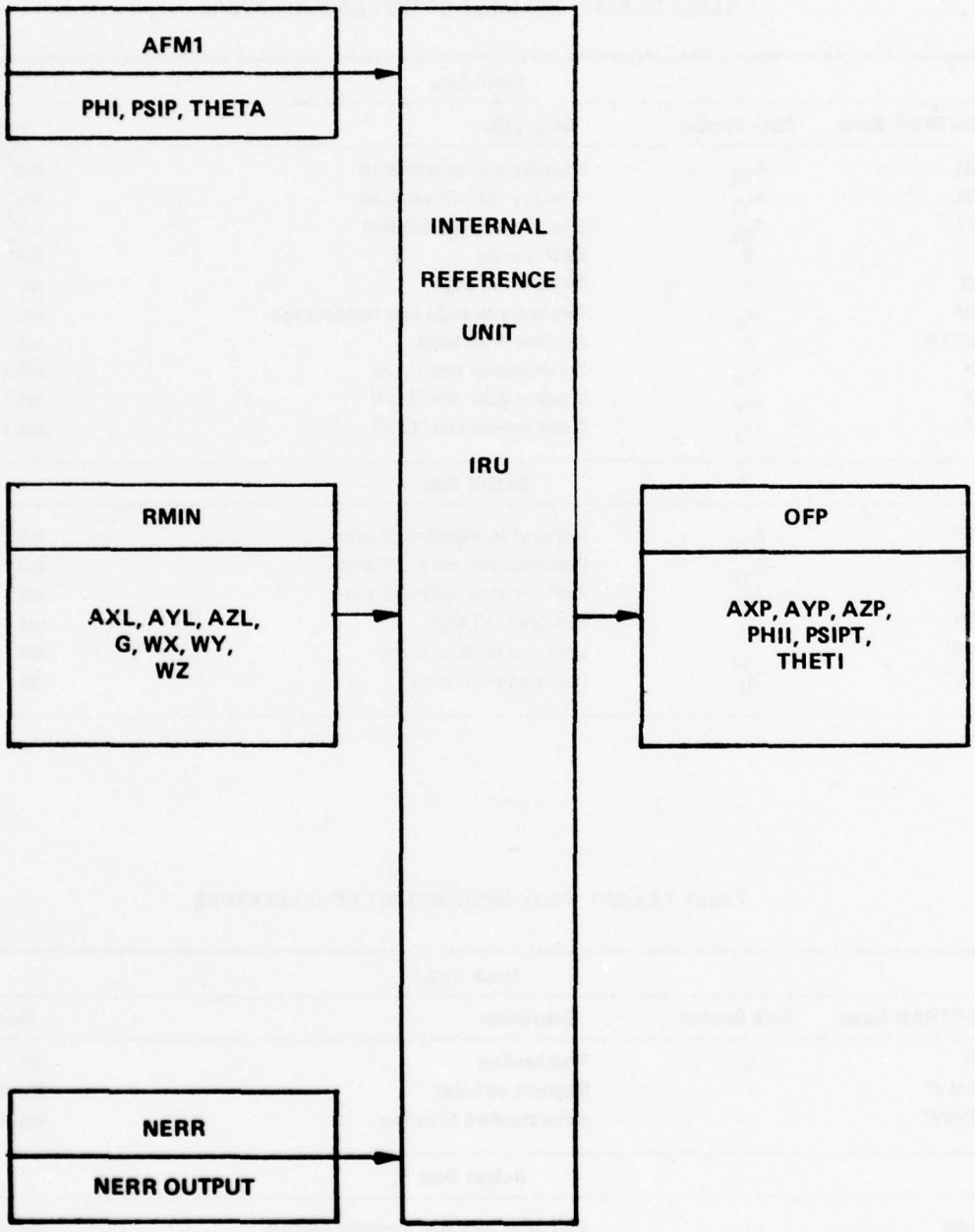


Figure 2.6.2.15-1. IRU input/output block diagram.

TABLE 2.6.2.15-1. IRU INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
Input Data			
AXL	A_{XL}	X locally level acceleration	ft/s^2
AYL	A_{YL}	Y locally level acceleration	ft/s^2
AZL	A_{ZL}	Z locally level acceleration	ft/s^2
G	g	Earth gravity	ft/s^2
PHI	ϕ	Airplane roll angle	rad
PSIP	ψ_p	Airplane yaw angle plus wander angle	rad
THETA	θ	Airplane pitch angle	rad
WX	ω_x	X axis angular rate (LL/I)	rad/s
WY	ω_y	Y axis angular rate (LL/I)	rad/s
WZ	ω_z	Z axis angular rate (LL/I)	rad/s
Output Data			
AXP	A_{xp}	Platform acceleration, X axis	ft/s^2
AYP	A_{yp}	Platform acceleration, Y axis	ft/s^2
AZP	A_{zp}	Platform acceleration, Z axis	ft/s^2
PHII	ϕ_i	Indicated roll angle	rad
PSIPI	ψ_{pi}	Indicated azimuth angle	rad
THETI	θ_i	Indicated pitch angle	rad

TABLE 2.6.2.16-1. FLUX INPUT/OUTPUT NOMENCLATURE

FORTRAN Name	Math Symbol	Description	Units
Input Data			
PSI	ψ	True heading	rad
PSIMV*	–	Magnetic variation	degree
SIGMV*	–	Noise standard deviation	degree
Output Data			
PSIMI	ψ_{mi}	Flux valve indicated magnetic heading	degree

* User input for initialization only.

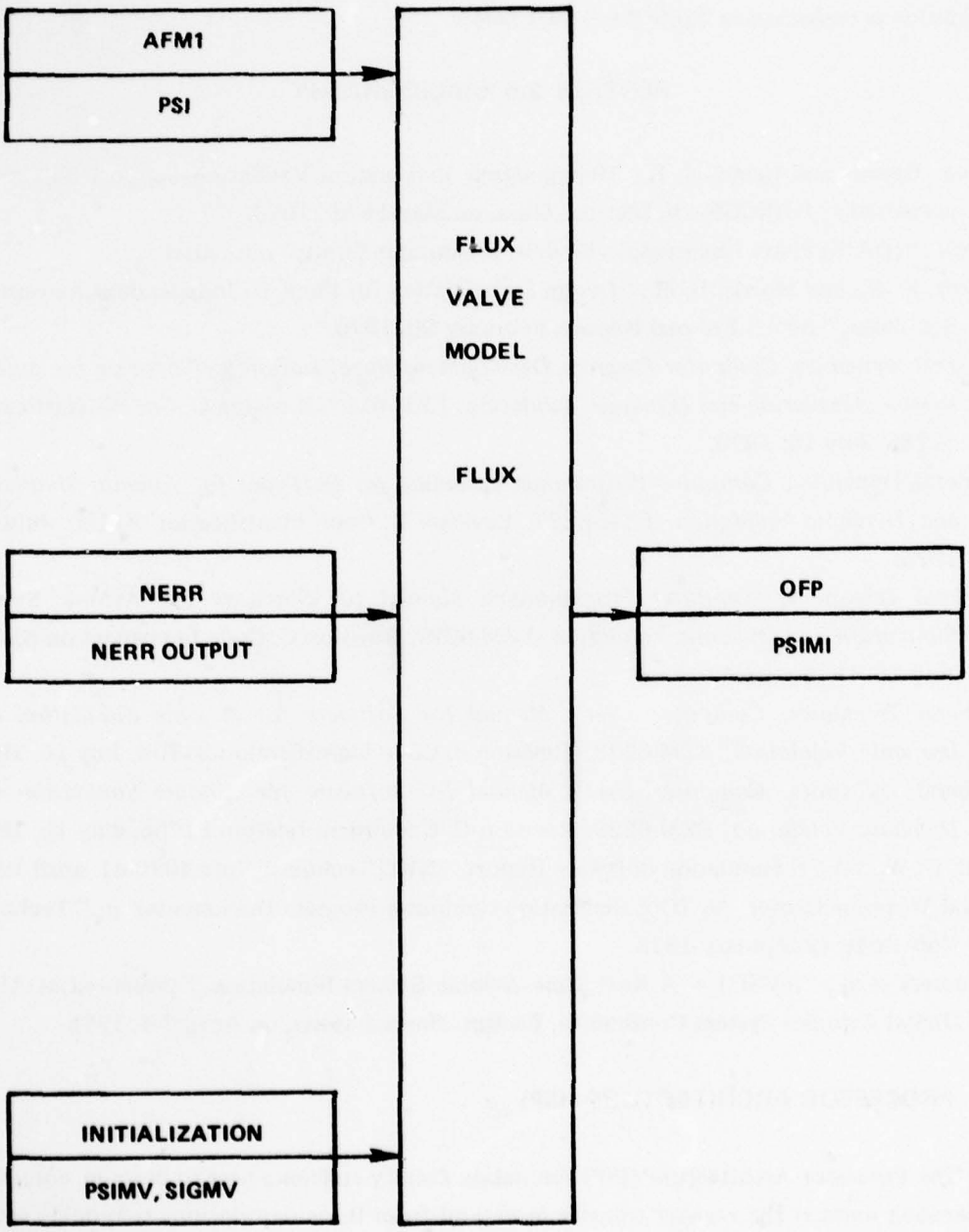


Figure 2.6.2.16-1. FLUX input/output block diagram.

which is routed to the navigation model NAV. The source model of each input is listed on the left while the disposition model is listed on the right. Additional nomenclature definition is contained in Table 2.6.2.16-1 below.

SECTION 2.6 BIBLIOGRAPHY

Allen, Byron, and Clema, J. K., "Independent Verification/Validation Support Software," presented at NAECON-76, Dayton, Ohio, on May 18-20, 1976.

Anon., "(DAIS) Phase I Simulation Models, Preliminary Draft," not dated.

Fissor, S. N., and Morris, D. M., "Design Specification for the F-16 Independent Assessment Simulator," AFAL Internal Report, February 20, 1976.

General Dynamics, *Computer Program Development Specification for Software for Avionic System Simulation and Dynamic Validation*, FZM-6197, Revision C, Code Identification 81755, July 15, 1976.

General Dynamics, *Computer Programmer's Manual for Software for Avionic Simulation and Dynamic Validation*, FZM-6227, Revision B, Code Identification 81755, July 16, 1975.

General Dynamics, *Computer Programmer's Manual for Software for Avionic System Simulation and Dynamic Validation*, FZM-6227, Revision C, Code Identification 81755, July 15, 1976.

General Dynamics, *Computer User's Manual for Software for Avionic Simulation and Dynamic Validation*, FZM-6228, Revision B, Code Identification 81755, July 16, 1975.

General Dynamics, *Computer User's Manual for Software for Avionic Simulation and Dynamic Validation*, FZM-6228, Revision C, Code Identification 81755, July 15, 1976.

Hall, C. W., "A-7E Simulation Software Report," NWC Technical Note 4070-41 April 1973.

Naval Weapons Center, "A-7C/E Simulation Computer Program Documentation," Technical Note 404-149, January 1973.

Summers et al., "AVSIM – A Real Time Avionic System Simulation," presented at AIAA Digital Avionics System Conference, Boston, Massachusetts, on April 2-4, 1975.

2.7 PROCESSOR ARCHITECTURE (ISP)

The Processor Architecture (ISP) simulation facility enables a user to describe computer processing units at the register transfer level, and from these descriptions to quickly set up interactive simulations of these processors. A language, CSL/ISP (Computer Simulation language, a dialect of Instruction Set Processor language), was developed as a dialect of ISPL from Carnegie-Mellon University. A compiler produces PDP-10 code from the CSL/ISP source and the user then runs this code from a control program which he constructs by

modifying a model, general purpose simulation control program. A methodology is included for creating simulations from manufacturers' instruction set descriptions.

From a description of the register transfers of a computer, the processor simulation program can produce a simulation of that computer. This program may be broken into three general areas. These include a formal language compiler with which to describe register transfer level processes, a code generator to produce DEC-10 code from the output of a compiler for that language, and a general purpose control program with which to drive the simulators produced from the compiler.

The compiler uses a set of register transfer operators, called XT-op's, to produce a set of pseudo instructions which can be interpreted by the DEC-10 assembler's (MACRO) facility. The macros which were written to produce DEC-10 code from these pseudo instructions produce optimized code, and they comprise a universal library which is searched during assembly.

Simulations are controlled by a program with which the user can interactively set registers and memory locations, load memory, set breakpoints, etc. The control program causes instructions in simulated memory to be executed by repetitively calling on the simulator to execute a single instruction.

The behavior of a processor is determined by the nature of its individual operations and the sequence in which those operations occur. This sequence is generally governed by a stored program, which resides in the memory of the computer and the set of interpretation rules which the processor applies to the program.

Although the above format is commonly used to describe digital computers, ISP does not limit the user to a particular type of description. Thus, ISP can be used to describe register transfer systems in general; digital computers are a subset of such systems which interpret an instruction set. Other devices, such as busses and device controllers, can also be described in ISP.

2.7.1 Specific Simulations

Three simulations were developed as test cases for this project and are presented in this section. *They are the INTEL 8080 microprocessor, the PDP-8 minicomputer, and the DAIS avionics processor.* The three simulations differ greatly in complexity, ranging from the simple PDP-8 to the quite complicated DAIS processor. The general process showing how to use the CSL/ISP system to produce and run a simulation is shown in Figure 1.7.1-1.

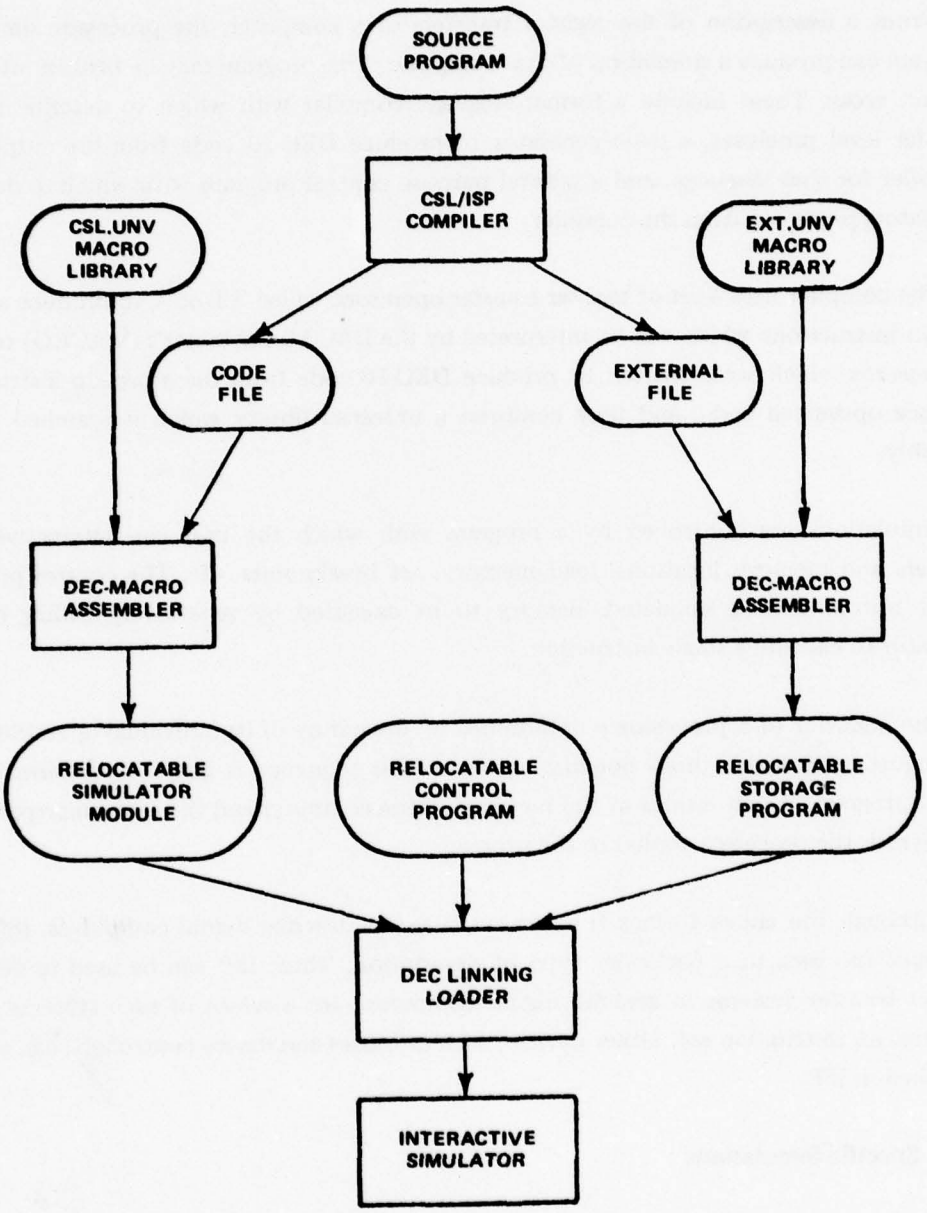


Figure 2.7.1-1. Procedure for generating a processor simulation.

2.7.1.1 The PDP-8 Simulation

The PDP-8 simulation was undertaken because of the simplicity of its instruction set (it has six memory reference instructions and two micro instructions) and because of its wide range of users. These reasons make it a useful demonstration example for the full use of the simulation facility, from the CSL/ISP description to the simulation itself.

The processor description includes the fetch cycle, the memory reference instructions and one micro instruction. The input/output micro instruction was not described. Some simple programs were loaded (from the output of the PAL10 assembler) and run. The PDP-10 to PDP-8 runtime ratio was about 90 to 1, well within the range of acceptable simulation times.

2.7.1.2 The INTEL 8080 Simulation

The development of a simulation module for the INTEL 8080 microprocessor through a CSL/ISP description was undertaken for two reasons. First of all, it was written in order to check operations with a large description (>200 instruction routines). Secondly, it was used to help test and debug the CSL/ISP compiler and the simulation control strategy. This second reason was particularly strong since a simulator of the INTEL 8080 was already available and results could be compared.

The simulator adheres to the description of the processor found in the INTEL 8080 manual (INT74) with three exceptions.

1. The undefined operation codes are trapped by the simulator as errors. The manual does not define the action of the processor for these operation codes.
2. One of the undefined operation codes was utilized as a break trap instruction. The operation code used is 20 octal.
3. All input and output for any channel is directed to the terminal.

The execution speed of the simulation was timed and found to be 100 to 1 compared to actual execution speed.

2.7.1.3 The DAIS Processor Simulation

This is the largest simulation implemented in ISP. The instruction set of the DAIS

processor, built for the Air Force by Westinghouse Electric Corporation, was described with CSL and subsequently simulated. Neither the bus control equipment nor the direct memory access eqipment was simulated.

The simulation extends beyond the machine described in the manual in two ways.

First, the interrupt system is modeled as described in the manual. However, the simulator will respond by calling a routine within the control program if any of the conditions for which interrupt responses are included occur when interrupts are disabled. Thus, the user can either elect to include code in his programs to detect interrupt conditions, or he may rely on the simulator/control program to inform him when these conditions occur. The following interrupt situations are handled by the simulator:

1. Illegal operation code (external routine OPERATION),
2. Boundary alignment error (external routine BOUNDARY),
3. Interval timer "A" interrupt (external routine TIMERA),
4. Interval timer "B" interrupt (external routine TIMEB), and
5. Processor memory protect (external routine PROTECTION).

Second, a branch trace facility was added to the simulation package. Any branch which alters the instruction counter (IC) is recorded in a 32-word table by the REMEMBER routine of the control array. The contents of the control array (J.ADDRESS) can be inspected by using SIX12; index word, J.INDEX, points to the next word of the array to be used. Entries are made in J.ADDRESS in cyclic fashion. Thus, the entry in the word indicated by the contents of J.INDEX is the *first* of the entries in the array to have been made. Each word of J.ADDRESS holds the location of the jump instruction in the upper half word, and the destination of the jump in the lower half word. The SIX12 typeout format types these values in the form UPPER, LOWER at the extreme right end of the line for each value typed.

Several instructions are simulated primarily by means of external routines written in DEC MACRO assembly language. All of this external code is included in two external modules. The two modules and the routines included in them are as follows:

1. Module FLOAT, source file FLOAT.MAC, relocatable file FLOAT, REL includes entry points:
   a. FA: floating add,
   b. FS: floating subract,

c. FNEG: floating negate,
d. FABS: floating absolute value,
e. FC: floating compare,
f. FM: floating multiply,
g. FD: floating divide;

2. Module FIXED, source file FIXED.MAC, relocatable file FIXED.REL includes the entry points:
   a. MPY64: fixed point double precision multiplication,
   b. DIV64: fixed point double precision division.

The simulator executes one instruction for each call by the control program. The first step in an instruction execution is to look for and to respond to any interrupts that may be pending if, and only if, interrupts are enabled. Following this, the simulator fetches the next instruction into the pair of instruction registers IRO and IR1. The bulk of the simulator performs instruction decoding, which begins after the instruction fetch is complete. The decoding consists of a 16-way decode of the high order operation code digit; each of the 16 elements in the range of this DECODE is a 16-way DECODE of the low order operation code digit.

Each simulation for each instruction assumes that IR0 contains the operation code portion of the instruction, and that IR1 contains the address portion of the instruction if there is one.

2.7.2 The CSL/ISP Language

CSL/ISP is a language for describing digital systems at the register transfer level, including descriptions of the computer memory, registers, instruction set, interrupt system and input/output system. A program in CSL/ISP consists of two parts, a description of the storage elements of the system, and a description of how the system operates on those storage elements.

2.7.2.1 Legal Characters

CSL/ISP input consists of a stream of ASCII seven-bit characters. The characters which the compiler accepts include the alphabetic characters A - Z and a - z. The compiler makes no distinction between upper and lower case alphabetic characters. In addition, the compiler accepts the numerals 0 - 9 and the characters [,], <, >, (,), $, ?, #, ', %, !, \, +, -, */, =, ^,@, :, <-, _ (left arrow, which is an underline character in some ASCII character sets), ., CR, LF, TAB, SPACE, and,. All other characters are illegal.

2.7.2.2 Identifiers

Identifiers are used as names of various computer components and as labels for statements and procedure declarations in a description. An identifier begins with an alphabetic character and is composed of any number and combination of alphabetic characters (both upper and lower case), numerals and the period, "." . Names must be distinct in their first six characters.

EXAMPLES:

X1, xi, Sixteen, One.Step, P296.1.

The names Pistep1 and PISTEP2 are recognized as the same identifier.

The compiler accepts an extension to an identifier which can be used to give more information about the symbol. The compiler treats the added information as a comment. The syntax for this type of comment is:

identifier/string of characters valid in an identifier

The "\" need not be preceded by an identifier, and all valid identifier characters which follow the "\" are ignored until a separator character is found. Thus, this construction can be used to insert comments into the middle of a line.

EXAMPLES:
```
\THIS.IS.A.COMMENT
\1 ! THIS IS A COMMENT TOO
SW.CON\CONSOLE.SWITCHES
```

Further examples are given in a later section.

2.7.2.3 Numbers

Numbers may appear in the input in binary, octal, decimal or hexadecimal form. Special prefix characters indicate the base of the number which the subsequent numerals designate. These prefix characters are:

'binary,

octal,
Digit decimal,
% hexadecimal.

The compiler converts the input representation into an internal binary number which cannot exceed $2^{18} - 1$.

EXAMPLES:

103 is 103_{10}
'1101 is 13_{10}
#72 is 58_{10}
%AF is 175_{10}

The number %7FFFF cannot be correctly compiled because it exceeds $2^{18} - 1$.

2.7.2.4 Program Structure

CSL/ISP is a block structured language; a CSL/ISP program is a labeled block which starts with a BEGIN and is terminated by an END. A CSL/ISP block may have a declaration section. A nontrivial CSL/ISP program will generally have a declaration section followed by an action section. The declaration section defines the elements upon which actions are performed. The form for a block is thus:

```
PDP8 := BEGIN
    optional declaration section
    action section
    END .
```

2.7.2.5 The Declaration Section

The declaration section permits the user to define the storage elements of the processor, namely, registers and memories. These storage declarations may represent the registers and memories of the computer being described, or they may be working storage for use in the simulation. In addition, external procedures, internal procedures, and macro declarations can be declared in the declaration section.

The declaration section occurs optionally as the first part of a block, and is delimited by the reserved words

(spell declare backwards). The declarations within the declaration section are separated by semicolons. Extra semicolons may appear, and the final declaration need not be followed by a semicolon.

2.7.2.5.1 Storage elements

The storage elements of a computer which can be described in CSL/ISP are registers and memories. The general declaration form for a register declaration is

name <bit range>

where *name* is any identifier, and *bit range* is either an empty field, a number, or a pair of numbers separated by a colon. Since the compiler generates a simulator which stores each register and each word of a memory in one DEC-10 word, register sizes may not exceed 36 bits, which is the size of a PDP-10 word.

A memory is an array of words all of which have the same bit specification. Its declaration has the form

memory [*word range*] < *bit range* >

where *memory* is an identifier, *word range* is a single number or a pair of numbers separated by a colon, and *bit range* is the same as that for registers.

Other computer components such as I/O ports and switch registers can also be declared as registers.

2.7.2.5.2 External declarations

To permit a control program and SIX12 (the BLISS debugging system) to access the storage of a simulated machine, memories and registers may be declared within an external declaration subsection, the form being

EXTERNAL *external declarations* LANRETXE .

An external subsection occurs as a declaration in a declaration section. Thus, it must be separated by semicolons from the other declarations. Memories and registers are declared in the external subsection just as they are declared outside it.

2.7.2.5.3 Overlay declarations

Registers and memories may be defined in terms of other memories or registers which have been previously declared. This allows one to define, for instance, the operation field of an instruction register as a register itself.

In a similar manner, overlays may be specified on memories. The constraints on memory overlays are more complicated than those on register overlays due to the DEC-10 assignment of arrays. When a memory is declared, its words are assigned one word at a time to consecutive DEC-10 words, so that, for instance, 4,096 twelve-bit words are assigned to 4,096 PDP-10 words. Memory overlays must be declared such that DEC-10 word boundary crossings are not implied by the declaration.

2.7.2.5.4 Procedure declarations

Procedures manipulate the contents of storage elements. Their declarations have one of the two forms:

name := *unlabeled statement*
name := *block*

where *name* is an identifier and is the name by which the procedure is called. The statement or block defines the actions of the procedure.

2.7.2.5.5 Macro declarations

The compiler has a macro facility which allows the user to substitute strings of characters for names which occur in his program. Macros may be declared with or without parameters. A macro declaration has one of the following two forms:

MACRO *name* = *string* $
MACRO *name* (*p1*,*p2*,. . . *pn*) = *string* $

where *name* is an identifier, *p1*,. . . ,*pn* are identifiers which name the parameters, and *string* is a string of characters which includes any character but the dollar sign, "$".

2.7.2.6 The Action Section

The action section consists of a group of parallel actions which are separated by the reserved word NEXT. A parallel action is a group of statements separated by semicolons. The interpretations of the sequence

statement 1 ; statement 2 ; statement 3
NEXT

statement 4 ; statement 5
NEXT
statement 6

is that statement 1, statement 2, and statement 3 are performed simultaneously. After their completion, statement 4 and statement 5 are performed. Finally, statement 6 is performed.

The semantic distinction between statements grouped for parallel execution and sequences of such groups has been reduced to a syntactic distinction in CSL/ISP. In a simulator produced by CSL/ISP, statements, whether separated by semicolons or NEXT reserved words, are executed sequentially.

The statements which can appear in the action section are of several types. Included among these elements are register transfer statements, conditional branching statements, jumps, procedure calls, return statements, and blocks.

2.7.2.6.1 Assignment statements

Transfers of data between storage elements are represented by assignment statements. The general form of an assignment statement is

target <- *computation*

where *target* is a set of contiguous bits in a storage element which is to receive the result of the *computation* indicated on the right side of the "<-". The computation is either an arithmetic or logical operation on bits of storage elements, or it is simply a reference to bits from a storage element. Any bit or contiguous group of bits in a register or memory word may be accessed for the reading or writing of information.

2.7.2.6.2 Operators

Data in storage elements may be modified as a result of various unary and binary operations on data in other storage elements. These operators are:

shift operators	The value of the expression is shifted,
NOT AND EQV OR XOR	Logical operators including the logical complement NOT,
unary + and − * / MOD	Unary operators,
+ −	Binary operators.

2.7.2.6.3 Nested expressions

The normal precedence for the operators may be altered by enclosing subexpressions in parentheses. Because CSL/ISP is intended for use in describing the actions of digital devices, the default operator precedence may differ from that in the normal higher level language such as FORTRAN.

2.7.2.6.4 Conditional statements

There are two different conditional statements in the CSL/ISP language, namely, the IF statement and the DECODE statement. Each of these statements makes use of a relational expression. The syntax and semantics of relational expressions are discussed in the following section. That section is followed by two sections which discuss the IF and DECODE statements.

2.7.2.6.5 Relational expressions

A relational expression has one of the two following forms:

X relational operator Y,

X.

Both X and Y in the above forms can be arbitrary expressions involving shift, logical, and arithmetic operators. The relational operators are:

LSS	less than,
LEQ	less than or equal to,
EQL	equal,
NEQ	not equal,
GTR	greater than,
GEQ	greater than or equal to.

2.7.2.6.6 The IF statement

The IF statement provides for conditional execution of any CSL/ISP action section. The form of the IF statement is

If relational expression = > action section ? .

The low order bit of the relational expression determines whether the action section which follows the "=>" is executed.

2.7.2.6.7 The DECODE statement

The DECODE statement selects one statement from a list of statements for execution. The form of the DECODE statement is

DECODE relational expression = > statement list ? .

The value of the relational expression serves as an index value to determine which one of the statements in the statement list is executed.

2.7.2.6.8 Flow of control statements

The flow of control in a CSL/ISP program is sequential. Statements are executed in the order written, even if separated by semicolons to imply parallel execution. The flow of control can be modified by three different statements:

1. Jump—jump to statement labeled by identifier,
2. Procedure-call statement, and
3. RETURN-return to point of call.

2.7.2.6.9 Blocks

A statement can also be a block. A frequent use of a block is to include an extensive action section as one alternative in the scope of a DECODE statement. A block includes an optional declaration section followed by an action section, and therefore has the form:

BEGIN
Optional declaration section
Action section
End

The part of the program in which a declaration holds is governed by the block in which it is declared.

2.7.2.6.10 Macro calls

A CSL macro can be declared with or without parameters. A macro declared with no parameters must be called with no parameters; a macro declared with parameters must be called with parameters. A macro is called by using its name. Parameters follow the name, enclosed in parentheses. The parameters are separated by commas. Extra parameters in a call

are ignored, and missing parameters are replaced by the null string in the expansion process. A macro declared with parameters must be called by

name ( )

to use the null string for all parameters. Recursive macro calls, including loops, are not allowed. The compiler detects an attempt to use macros recursively and reports the name (or names if a loop is involved) of the offending macro(s) in the error message.

2.7.2.7 The Control Program

A simple control program should rely on the SIX12 debugging package of the BLISS system for many of the facilities which it provides to a user. SIX12 can interact with the user to display and alter the values of global and own variables of BLISS and MACRO programs. Thus, SIX12 can be used to inspect memory and register contents of a simulated machine, alter those values when necessary, and in general, control both the debugging and use of a simulator. The normal functions of a simple control program are:

1. Call a loader module which is written specifically for that simulator; the loader is written to be able to read a file prepared by a cross-assembler linking-loader system which produces a program file for the simulated computer; the program file input often includes a starting address for execution, so that the loader will often set the instruction counter of the simulator as well as loading code into its memory.
2. Make periodic calls on SIX12 so that the user can control the simulation process.

2.7.2.8 The Register Transfer Operations

The register transfer operations which occur in the <input file name>.MAC output file of the CSL/ISP compiler and which define the executable code of the simulation module are known as XT-operations within the compiler. In the <input FILE name>.MAC file the general form for an XT-operation macro call is

operation (SRC1, SRC2, DEST)

where SRC1 and SRC2 are the sources for the operation, and DEST is the destination for the result of the operation involving SRC1 and SRC2. In some cases, the operands are optional or take special forms.

2.7.3 Processor Simulation

2.7.3.1 The Computer Simulation Language Compiler

The Computer Simulation Language (CSL) compiler is written in the *BLISS*

implementation language for the DEC-10. The compiler is organized into seven BLISS modules. In each module, there is one primary routine and a collection of supporting routines. The following table gives an overview of the compiler organization.

Module	Primary Routine(s)	Primary Function
CSGBL	main body	Compiler initialization and global declarations
CSPRS	PRSE	Lexically analyze and parse the source program
CSMAC	DEFMACRO & EXPMACRO	Accept and expand macro definitions
CSSEM	SEMANTICS	Perform code generation and semantic actions corresponding to terminal symbols
CSGEN	GENERATE	Perform code generation and semantic actions corresponding to syntactic reductions during parsing
CSOPT	OPTIM	Reorder the code and optimize the flow of control
CSOUT	OUTTABLES	Write out SYTABLE and STTABLE for later assembly

The CSL computer is a *two pass compiler*. During the first pass, compiler modules CSPRS, CSMAC, CSGEN, and CSSEM cooperate to parse the source program and translate it into an encoded version of the eventual output in the statement table, STTABLE. During the second pass, compiler module CSOPT reorders the code in STTABLE and optimizes the flow of control. Finally, module CSOUT writes out the contents of the symbol table (SYTABLE) and the statement table (STTABLE) for later assembly by the DEC assembler MACRO.

2.7.3.1.1 The syntax graph

Parsing is done top-down and is table driven from a syntax graph. The graph is reduced by factoring common strings of leading symbols in a set of alternatives for a production. Recursive productions are recognized for further simplification and modes where values and links are identical are eliminated.

2.7.3.1.2 The parser

The routines which parse CSL are in the module CSPRS. Parsing is accomplished by an implementation of a nonrecursive algorithm. The algorithm has been modified to eliminate the pointer stack and to include recognition of missing right hand delimiters. The parsing process deals with two classes of objects: terminal symbols and recognized productions. Terminal symbols are handled by the compiler module CSSEM and recognized productions are operated upon by the compiler module CSGEN.

2.7.3.1.3 The syntax analyser

The routine PRSE performs the task of parsing the CSL source and of making calls to the semantic routines SEMANTICS and GENERATE in order to produce code. In order to analyze the input, PRSE calls on the routine SCAN which fetches and analyzes the next lexical unit, or lexeme, in the input. It does this by "sliding" a "window" through the source in order to set up the new current lexeme. In addition, this window positions itself over the last two lexemes analyzed and over the next two lexemes past the current lexeme. New entries conceptually are entered into the window from the right and disappear from the left. A new entry is made when SCAN calls the routine GETLEX. GETLEX determines the type of lexeme coming up either from its first character or from the value of the flag, FLAGSTRING, which is set by the semantics when a macro definition is arriving on the input. GETLEX then calls IDENT, NUMBR, OPERATOR or DERMACRO to fetch the lexeme and set the window entry.

After the window has been advanced, the parser compares the value of the current lexeme in the window with the value of the current node in the syntax graph. If they match and certain other conditions are satisfied, the parser calls the semantic routines to produce code. Otherwise a new node is selected and the comparisons continue until the proper node is located.

2.7.3.1.4 Code generation

The code generation process is controlled by the contents of various stacks, flags, and other variables. The contents of these control variables are set by the code generation routines and, in turn, control the actions of those routines.

All of the codes produced by the compiler are generated by calls to routine STATE of compiler module CSGEN. Routine STATE responds to calls by placing an encoded from of the eventual output in the statement table, STTABLE.

2.7.3.1.5 Compiler output

Except for the listing file (<input file name>.LST, produced by compiler module CSPRS), all output from the CSL compiler is produced by compiler module CSOUT. The output consists of two files when the source file contains EXTERNAL declarations and one file when no EXTERNAL declaration occurs. Both output files are intended to be assembled by the DEC assembler MACRO; there is a macro library appropriate to each output file.

The primary output file (<input file name>.MAC) contains the internal storage declarations and executable code for the eventual simulator; it should be assembled using the macro library CSL.UNV. The file produced when EXTERNAL declarations occur is <input file name>.EXT; it contains storage allocation and ENTRY definitions for the EXTERNAL variables and should be assembled using macro library EXT.UNV.

Compiler module CSOUT writes both of the above output files. The variable EXTRN (one of the variables in the control array CSVAR) indicates when an external file is necessary. CSOUT initially goes through the symbol table, SYTABLE, and writes the file <input file name>.EXT and the storage allocation part of file <input file name>.MAC. CSOUT then goes through the statement table, STTABLE, and writes the executable portion of the <input file name>.MAC file.

2.7.3.1.6 Error detection

The CSL compiler can recognize many such errors and informs the programmer about missing or extra program elements. The errors detected by the compiler fall into two classes, syntactic and semantic errors. The following two sections discuss these two classes.

1. Syntactic Error Detection—a syntax error occurs when the structure of the source program does not conform to the structure of the CSL language as defined by the syntax of that language expressed in the file CSL10.BNF. The most common source of syntax errors is a missing terminal symbol such as ";" or LANRETXE. The parsing process used by the compiler is top-down, so that a definite goal symbol is not always known to the parser. Hence, many errors of this type result, after a long futile search, in the message "parse failure." On the other hand, the CSL language contains many terminal symbols which must always occur in pairs; the parser can detect the absence of the second member of such pairs and point out the place in the program that it expects such a symbol to occur for a successful parse. The parser is often wrong about the place in the program, but it is never wrong about the absence of the matching terminal symbol.

In some cases, the compiler can determine that a ";" or NEXT symbol has been misused. Also, when the compiler expects an operator (eg., +, -, *, etc.), and finds something it does not recognize as a valid operator, it reports a syntax error.

A symbol which begins with a digit and contains non-digits before a separator symbol occurs will be reported as a number error.

2. Semantic Error Detection—a program can be syntactically correct and still contain errors which the compiler can detect. The following describes the different semantic errors which the compiler can detect.

The compiler reports a fatal semantic error when it detects a reference to a variable in an expression which has not been declared at the point where it is referenced. When an undeclared variable occurs as a jump destination, the compiler assumes that it refers to an undeclared external procedure. The compiler lists all variables of this type at the end of the listing file (<input file name>.LST) and on the controlling terminal.

The compiler reports a multiple declaration when it finds more than one declaration for the same identifier within the same block scope. A given identifier can be declared as an EXTERNAL variable only once in a program. Moreover, that same identifier may not be declared as a local variable anywhere in the program.

The CSL language permits a user to declare and access array variables. The syntax for both the declaration and access requires that an expression of some sort enclosed between matching brackets, "[. . .]", follow each instance of such a variable. The compiler reports a fatal semantic error when it finds a variable that was not declared as an array variable accessed with an expression enclosed in brackets. The compiler will report a reference to an undeclared word if an array reference occurs with a constant subscript and the constant used does not address a word declared for that array.

The syntax of CSL requires that all references to bits within a word or expression be composed of literal constants. The compiler checks to see:

1. That all of the bits referred to in a reference are declared for the variable to which the bit expression (<. . .>);
2. That the bit numbers in a bit reference occur in the correct order; thus, if "A" is declared "A <7: 0>" and referenced "A <0: 2>", the compiler will report a non-fatal "bit access reversed" error;
3. The number of bits and words declared for an overlay variable.

2.7.3.1.7 Compiler modification

The compiler for the CSL language is based on a table of the syntax of the language and two auxiliary files prepared by the syntax analysis program GRPGEN.SAV.

One of these auxiliary files, CSL10.EQV, permits the user GRPGEN to define equivalent names for terminal symbols and production names. The second auxiliary file, CSL10.NEG, permits the user to associate that which appears to be terminal symbols of the grammar with items which his lexical scanner will recognize in the source program. In CSL, they are NUMBER, IDENTIFIER, and STRING.

In general, the compiler can be modified in a fairly straightforward manner. If a syntax change is made that involves new terminal symbols and new production names, ordinarily, only modules CSSEM and CSGEN need be altered. Occasionally a syntactic test may have to be inserted into a parser to cause a search to proceed down a different path in the syntax graph if some condition is not met.

The routines SEMANTICS in CSSEM and GENERATE in CSGEN are simply long SELECT statements. In SEMANTICS, the terminal name passed is compared with a list of terminals. If a new terminal is introduced into the syntax, its equivalent name must be entered into the list. In addition, a call must be entered to a semantic routine for that terminal. This routine is written to perform the table manipulation and code generation.

The routine GENERATE is also a long SELECT statement. The name passed to it is of a production. Therefore, new production names which require semantic actions must be entered into this list and a call to the proper code generating routine must be made. Ordinarily, a new routine must be written for each new production name.

2.7.3.2 Code Generation

The output file(s) of the CSL compiler must be assembled with the DEC assembler MACRO before linking them with a control program for use as a simulator. There are two basic parts of the output:

1. Storage allocation, and
2. Executable code.

These two parts are discussed in the following sections.

2.7.3.2.1 Storage allocation

There are three classes of variables for which storage and names are required in a simulation module:

1. EXTERNAL symbols,
2. Internal variables, and
3. Compiler generated variables and constants.

The CSL language requires all EXTERNAL variables to have unique names, since these names will eventually acquire definition through the use of the DEC LINK program. However, the block structure of the language permits multiple use of the names for internal variables—those not declared EXTERNAL. The compiler generates requirements for temporary variables of two types. Since nonexternal names can be used more than once, and since the MACRO assembler does not have a block structure capability, all internal and compiler generated variables (and constants) are assigned compiler generated names for use throughout the simulation module. Two different types of names are assigned to internal variables:

1. User declared variables, constants, and pointer values are assigned names of the form Tnn, where "nn" is a decimal number,
2. Compiler generated temporary variables (TEMP's and FLAG's) are assigned names of the form %nn, where "nn" is a decimal number.

For all but the array variables, the storage allocated by a storage allocating macro is one word. For arrays, a word that contains the size of the array is allocated, followed by the number of words declared for the array. The word that contains the size of the array is used throughout the simulation to check for references outside the declared array.

The REGISTER, LREGISTER, EXREGISTER, MEMORY, LMEMORY, and EXMEMORY macros define an assembly time symbol of the form .<symbol name>, which contains the number of bits intended for the variable. A symbol of the same form with value zero is defined for CONSTANT, TEMP, and POINTER variables. (The value of the bit length assembly time variables throughout the assembly process are generated by LENGTH macro of the CSL compiler.)

Since symbols within MACRO are unique only to six characters, EXTERNAL symbols should have names unique to five characters (plus the prefix ".") so that correct length information can be accessed during assembly of the executable code.

2.7.3.2.2 Executable code

The allocation macros use DEC-10 registers for temporary and flag variables as long as they are available. The code generation macros optimize the code they produce when temporary and flag variables have been allocated DEC-10 registers. Variables having lengths of zero are accessed by word accessing rather than by byte pointers. Other than these two steps, the code generation process is straightforward. The pointers generated by the compiler all expect the address of the variable to be in register PREG, which is register 17 (Octal) in both the compiler and the macros. Accesses that use compiler generated pointers are accomplished by loading this register before using the pointer. In other cases, a pointer is generated as a literal during the assembly process.

2.7.3.3 Simulation

This section covers the considerations involved in utilizing a simulation module that has been generated through the CSL.ISP compiler. A simulation control program is discussed and specific simulations are described.

2.7.3.3.1 Simulation control considerations

The CSL/ISP compiler is used to process a register transfer level description of a digital processor into an executable simulation module. Since there are no input and output functions built into the CSL/ISP language, another module written in a language with input/output support is needed to interface the simulator with the user. Also, since the normal reason for use of this type of simulation is software or hardware evaluation, it is necessary to have some debugging-type controls over the simulator. Such controls include a complete display capability for all simulated memory and registers, single-step execution, and breakpoints.

2.7.3.3.2 Processor description methodology

The description method was chosen to provide the fastest execution time for a simulation, while providing all the desired debugging control facilities. An underlying feature of the method is that the simulator is a single-step routine. Each time the simulator is invoked, it simulates a single instruction and returns. In this way, the control module can manage the simulation on an instruction by instruction level. There are a number of reasons for this choice. First, it was decided that the execution control loop should be invariant over different simulators. Thus, it should neither be necessary nor desirable that the user code it

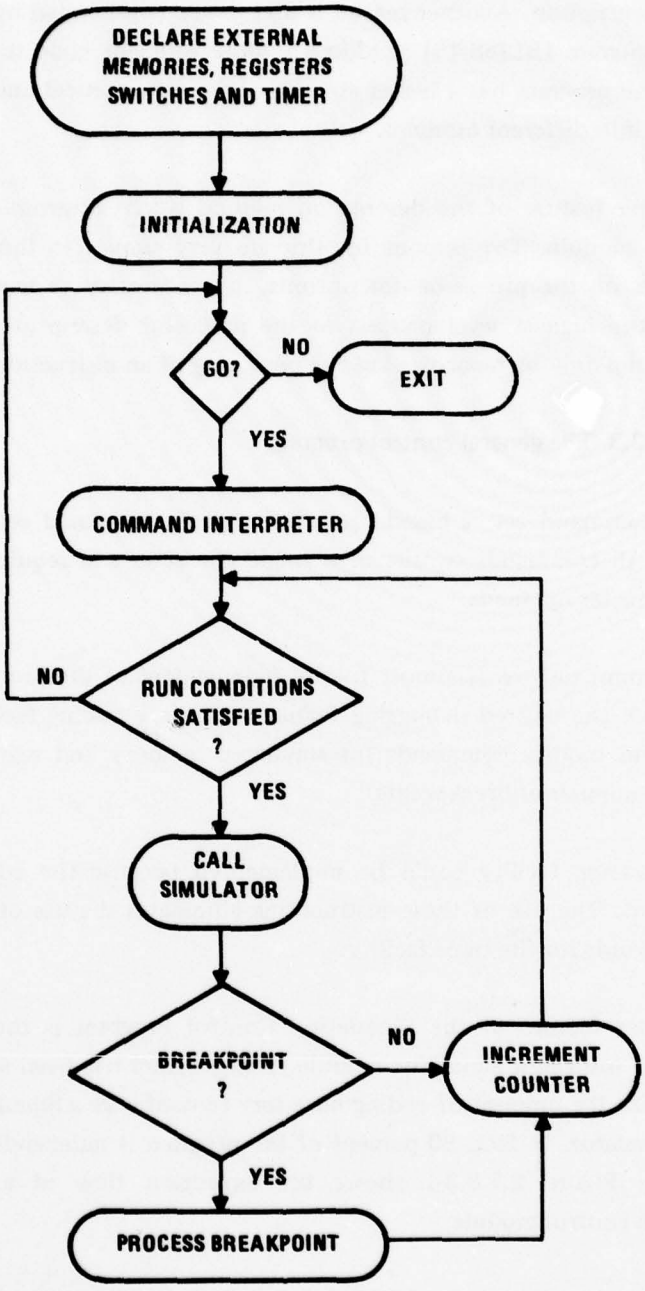


Figure 2.7.3.3-1. Simulation control program execution flow.

into his description. Another reason is that the code generated by the compiler used for the control program (BLISS-10) produces a more efficient code than the CSL/ISP compiler. Finally, the program has a better structure when the control and simulations functions are separated into different modules.

Another feature of the description method is that interrupts are to be handled in the simulator module. The reasons for this are very similar to those above. With these two constraints on the processor descriptions, it is necessary to have the fetch and execution cycle as the highest level process in the processor description. It should be noted that interrupt can only be recognized at the beginning of an instruction execution cycle.

2.7.3.3.3 The general control program

The command set is based upon the console command set for the DAIS Intelligent Console. All commands consist of a single character and require zero, or two numeric or single character operands.

The command set is almost totally independent of the processor being simulated and provides all the desired debugging features except a tracing facility. It supports extensive display and modify commands for simulated memory and registers, and can support an unlimited number of breakpoints.

The tracing facility could be implemented because the compiler generates fullword instructions. The use of these instructions eliminates the use of flag bits in the simulated memory words for the trace facility.

Another feature of the Simulation Control Program is the ease in which it can be configured for a new simulator module. The program has been structured in such a way as to minimize the amount of coding necessary to configure a Simulation Control Program for a new simulator. In fact, 80 percent of the program is independent of the processor being simulated. Figure 2.7.3.3-1 shows the execution flow of a simulation utilizing this simulation control module.

SECTION 2.7 BIBLIOGRAPHY

AFAL, "Processor Architecture Simulation," Contract No. F33615-74-C-1112, Final Report, June 10-30, 1976.

Graham, M. L., Peterson, L. J., and Rude, M., *User's Guide to CSL/ISP*, Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, Illinois, June 1, 1976.

2.8 COMMUNICATIONS SYSTEMS EVALUATION LABORATORY

2.8.1 Introduction

A framework in which to relate the capabilities of the Communications System Evaluation Laboratory (CSEL) is provided in Figure 2.8.1-1. This figure depicts a standard block diagram of a general communication system consisting of a transmitter (information source, encoder, modulator, up converter), a channel, and a receiver (down converter, demodulator, decoder, and information sink). Of primary interest in this discussion are the carrier systems involved; thus, the appropriate carrier frequency associated with any of the links of the system has been indicated in the block diagram. For completeness, it has been assumed that appropriate antenna systems are associated with the transmitting and receiving terminals.

The channel of the communication system in Figure 2.8.1-1 can take one of the two forms shown in Figure 2.8.1-2. The first of these involves a single link between the

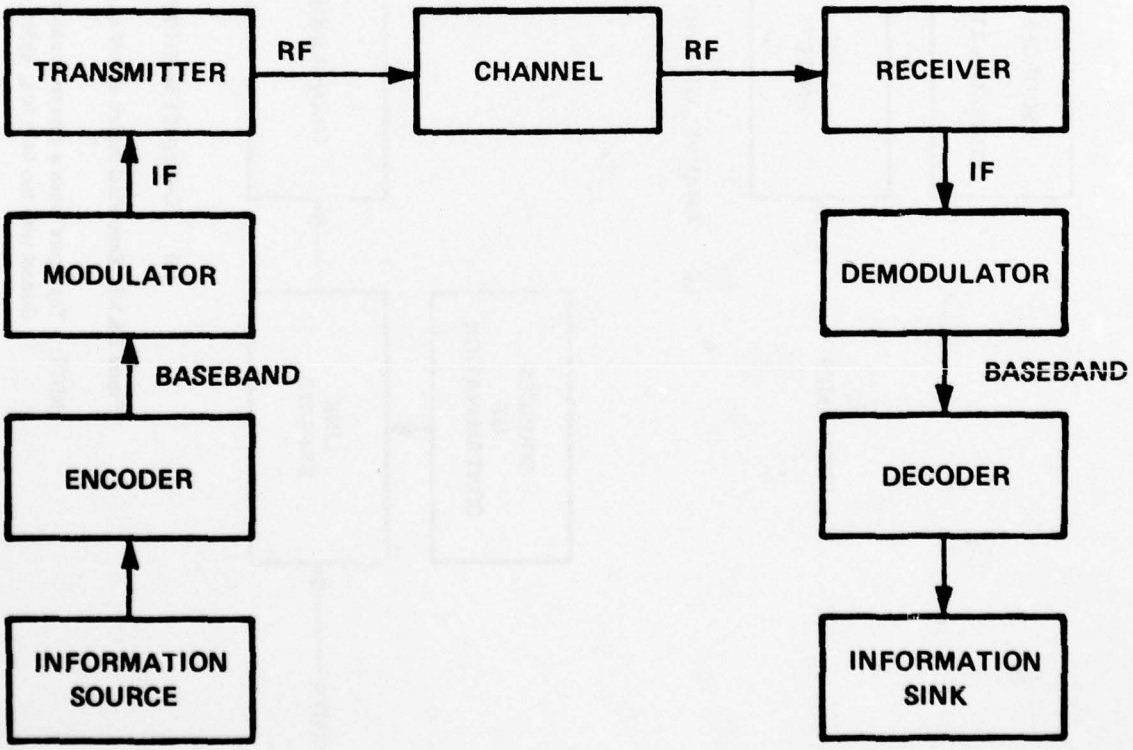
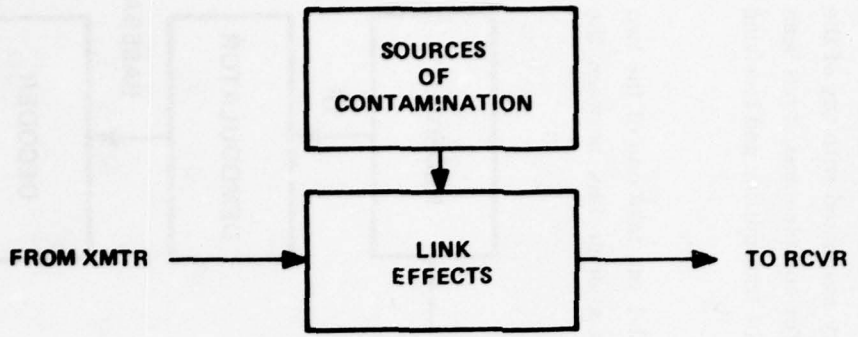
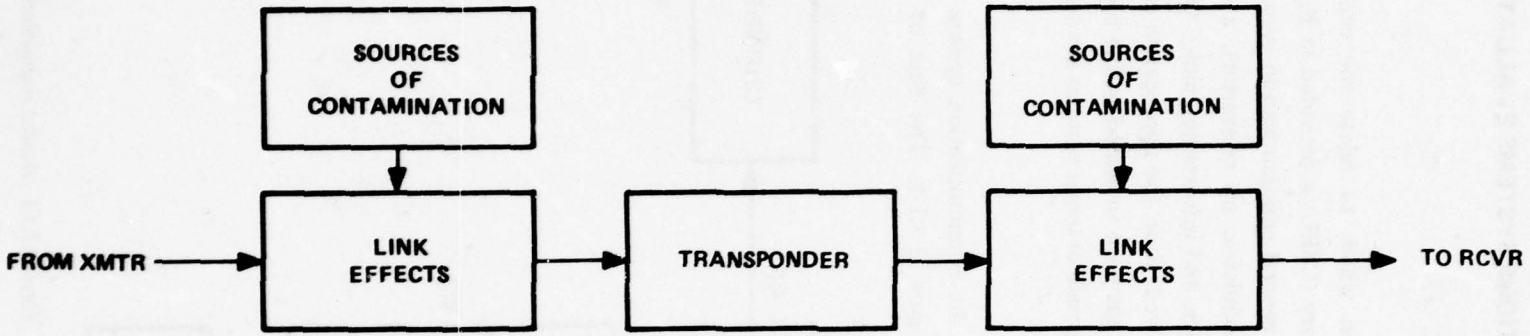


Figure 2.8.1-1. General communication system block diagram.



(a) CHANNEL WITH NO TRANSPONDER



(b) CHANNEL WITH TRANSPONDER

Figure 2.8.1-2. Conceptual models of two types of communication channels.

NOTE: Top figure shows a channel consisting of a single link; bottom figure, a channel with two links (e.g., uplink and downlink).

transmitter and receiver terminals while the second consists of two links separated by a transponder.

In both cases, the sources of contamination are assumed to produce signals which combine with the information-bearing signal and degrade its reception. In addition to receiver front-end noise, the contaminating signals of primary interest are jamming signals and cochannel and interchannel interference. Likewise, channel effects which degrade reception are fading, multipath, and doppler shifts. The block diagram of Figure 2.8.1-2 is intended to show that the channel effects act together with the contamination sources to corrupt the desired signal. In what follows, these factors are referred to jointly as environmental (i.e., electronmagnetic) effects.

The primary role of CSEL is to furnish the means of evaluating communication system performance degradation because of these environmental effects. CSEL provides a user with the capability of simulating the channel at his communication system terminal—with hardware and in the appropriate frequency band, thus allowing him to create an RF signal environment suitable to his application. With respect to Figure 2.8.1-2, the facility includes hardware to generate known contaminating signals, to combine them with an information-bearing signal and to subject the signals to various channel effects. In addition, CSEL allows the user either to simulate the transponder shown in the figure or, in certain cases, to employ an actual (satellite) transponder.

As shown in Figure 2.8.1-1, the significant general-purpose capability of CSEL is the creation of realistic communication channels, which, in turn, provides the user the ability to test various communication concepts as well as to test actual communication gear. It is implied that the equipment required to outfit a complete laboratory communication system—antennas, transceivers, modems, and codecs—must, in general, be user-supplied, although CSEL does make available to users a variety of such gear.

Despite the general nature of CSEL previously described, the current thrust of CSEL is directed at studying jamming and antijamming techniques for application to satellite communication links. In addition, CSEL makes available the means of instrumenting antijamming measures including programmed frequency hopping, Thus, in Figure 2.8.1-2, the RF input signals can be thought of as spread-spectrum signals whose structure is user-controlled.

An additional factor which currently limits the scope of potential application for CSEL is its present emphasis on digital communications systems (although purely analog systems

are not ruled out of consideration). This is indicated not only by the presence of coders in Figure 2.8.1-1, but also by the fact that the hardware available for transponder simulation consists of a digital signal processor. Thus, system transponders, a block diagram of which is shown in Figure 2.8.1-3, appear to function as coders even if the only operation they perform is to reconstitute a digital data stream.

2.8.2 Signal and Interference Generator

The task of channel simulation in the Communication Systems Evaluation Laboratory is performed by two pieces of equipment: the Signal and Interference Generator (SIG), sometimes referred to as the K-band Terminal Simulator; and the Programmable Data Terminal (PDT). Discussion of the latter device, used primarily for transponder, simulation, is postponed to the next section.

An overall block diagram of the SIG is shown in Figure 2.8.2-1. This system, developed by Computer Sciences Corporation, is capable of modeling various types of RF links and the

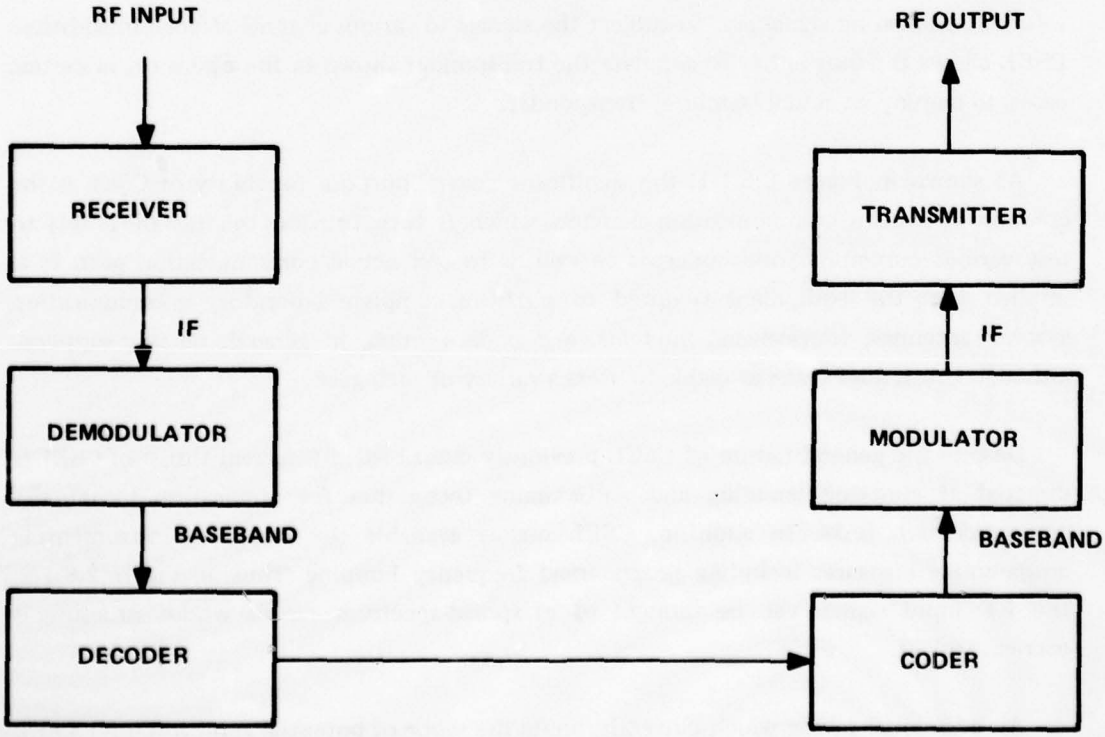
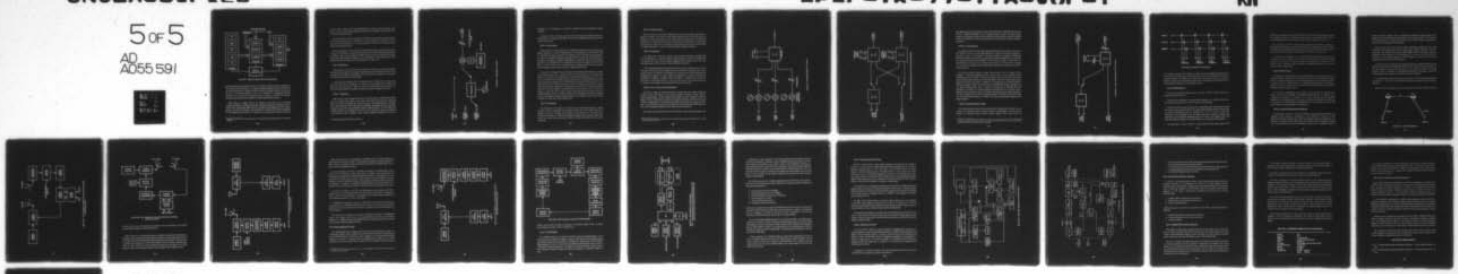


Figure 2.8.1-3. Transponder block diagram.

AD-A055 591 RESEARCH TRIANGLE INST RESEARCH TRIANGLE PARK N C F/G 1/3
AFAL SIMULATION FACILITY/CAPABILITY MANUAL. VOLUME I. EXECUTIVE--ETC(U)
JUN 77 R A WHISNANT, W H RUEDGER, R L EARP F33615-76-C-1308
UNCLASSIFIED AFAL-TR-77-118-VOL-1 NL

5 OF 5
AD
A055 591



END
DATE
FILMED
8-78
DDC

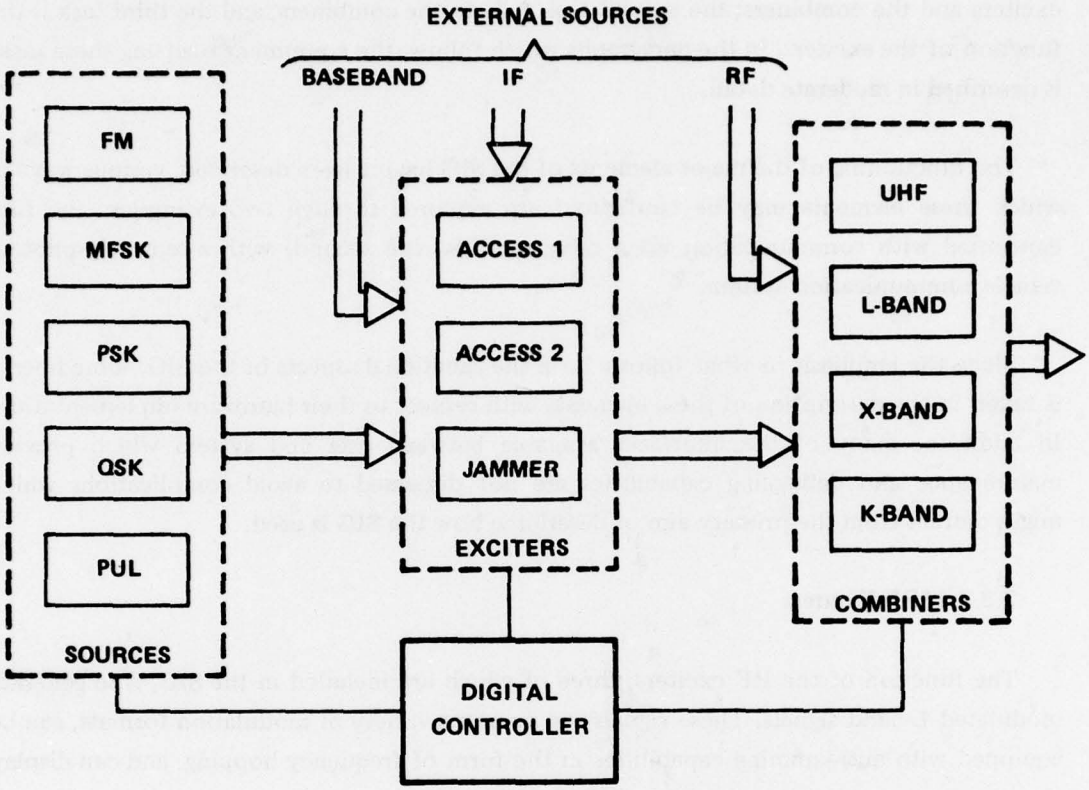


Figure 2.8.2-1. Overall block diagram in signal and interface generator.

signal environments associated with them. As the figure indicates, the SIG is composed of four primary functional elements: (1) analog and digital baseband sources; (2) RF exciters; (3) RF combiners; and (4) a digital controller. In general, the digital controller, operating on the basis of commands supplied by the user, is responsible for configuring the other elements so as to model the communication links of interest. In addition, the controller exerts real-time control over the operation of these elements during a test run.

With reference to Figure 2.8.1-2, the purpose of the SIG is: (1) to generate contaminating signals including jamming and cochannel/inter-channel interference; (2) to combine the contaminating signal with the information-bearing signal; and (3) to subject signals to link effects, including fading and doppler.* In general, the first task is performed by the baseband sources acting in conjunction with up-converters located in both the

* Multipath, mentioned in the preceding section, is not instrumented, although plans to do so have been discussed.

exciters and the combiners; the second task falls to the combiners; and the third task is the function of the exciters. In the paragraphs which follow, the equipment realizing these tasks is described in moderate detail.

The functioning of the major elements of the SIG having been described, various ways in which these elements may be configured are explored through two examples: the first concerned with communication via a relay satellite; the second, with a remotely-piloted-vehicle communication system.

Since the emphasis in what follows is on the functional aspects of the SIG, some liberty is taken in the description of these elements with respect to their hardware implementation. In addition, many of the interfaces available between user and system which provide maintenance and debugging capabilities are not discussed to avoid complications which might detract from the primary aim of describing how the SIG is used.

2.8.2.1 RF Exciters

The function of the RF exciters, three of which are included in the SIG, is to generate modulated L-band signals. These signals can possess a variety of modulation formats, can be equipped with anti-jamming capabilities in the form of frequency hopping, and can display time-varying power levels to simulate fading.

A functional block diagram of the RF exciters, shown in Figure 2.8.2.1-1, displays the three basic functions of modulation, up-conversion, and attenuation they perform. The exciters accept inputs from a number of sources, some internal and some external to the SIG, as the figure indicates. In general, external sources must be supplied by the user.

2.8.2.1.1 Modulators

With respect to their modulation function, the exciters utilize a 560 MHz signal generated by a phase-locked oscillator as the basic carrier. With some restrictions mentioned below, an exciter can modulate this carrier in any one of the following ways: continuous wave (CW); frequency (FM); phase shift keying (PSK); quadraphase shift keying (QSK); multiple frequency shift keying (MFSK); and pulse modulation (PUL).* In this context, CW modulation refers to no modulation; the output of the modulator is simply the 560 MHz carrier. Moreover, PUL modulation refers to pulse amplitude modulation. The type of

* The acronyms correspond to CSC documentation.

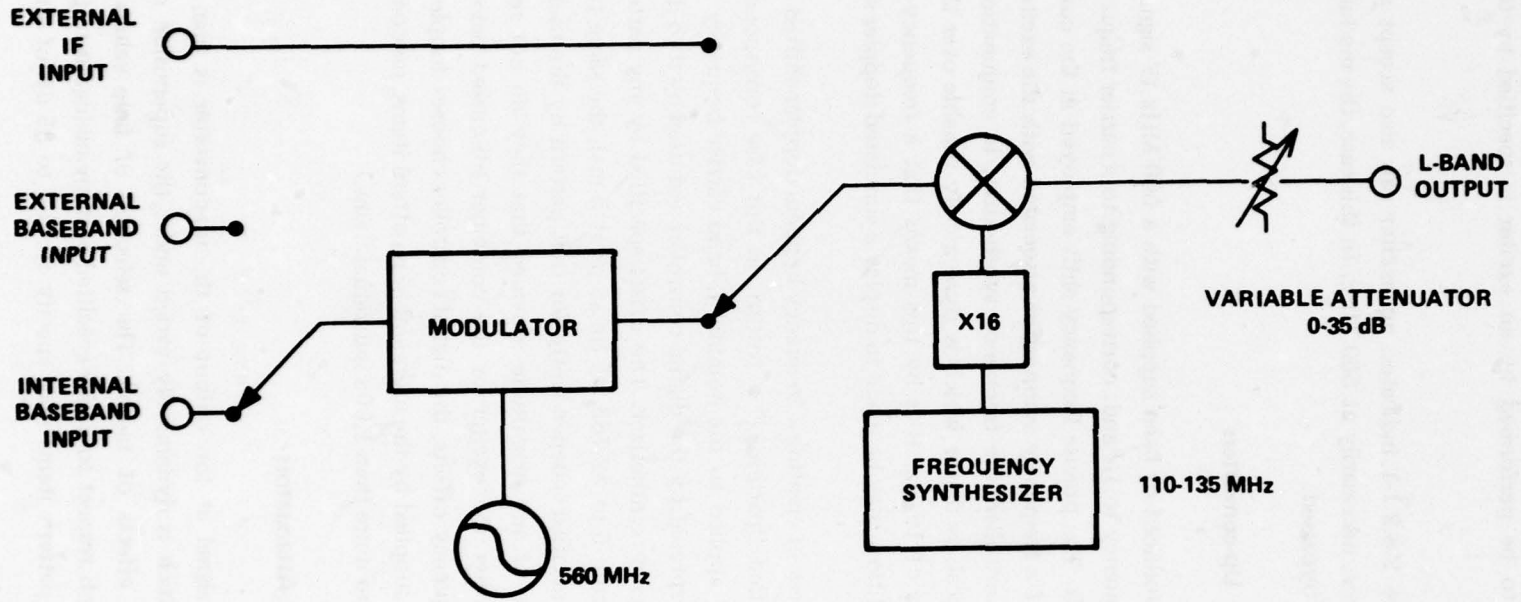


Figure 2.8.2.1-1. Block diagram of RF exciters.

modulation to be performed by an exciter is specified by the user through the digital controller.

As Figure 2.8.2.1-1 indicates, an exciter can also accept an IF signal input from an external source, necessarily at 560 MHz. In this case, the modulation portion of the exciter is effectively bypassed.

2.8.2.1.2 Up-converters

Having produced or been supplied with a 560 MHz IF signal, the exciter translates this signal in frequency to L-band, corresponding to a carrier frequency in the band from 1,200 to 1,600 MHz. The precise frequency shift employed in the conversion varies according to the output of a frequency synthesizer associated with the exciter. Under the supervision of the digital controller, the frequency synthesizer, in conjunction with an X16 multiplier, provides a local oscillator which is dynamically tunable over the range of 1,760 to 2,160 MHz in steps of 16 Hz. It is by this means that a frequency-hopped signal is produced which, in addition, may be made to display a simulated doppler shift.

The process of creating a frequency-hopped, doppler-shifted signal takes place according to user-specified "patterns," a pattern in this case corresponding to a list of frequency offsets to be applied to the nominal L-band carrier frequency assigned to an exciter. Patterns are interpreted by the digital controller and used by it to dynamically vary the output of the frequency synthesizer. The offset specified by any pattern item may persist in the synthesizer from 5 to 32,765 ms (in steps at 5 ms), the same time for all items in a given pattern but specified independently for that pattern by the user. The values of the pattern items may be of any magnitude provided that they do not result in an out-of-band frequency in either the exciter or the combiner (discussed below) into which it feeds. In selecting frequency offsets, the digital controller chooses doppler pattern items sequentially from the list supplied by the user, and hop pattern items, randomly. In both cases, a pattern is limited to no more than 1,008 individual items.

2.8.2.1.3 Attenuators

The RF signal at the output of the up-converter is then passed through a variable attenuator which is dynamically varied under the supervision of the digital controller to simulate the effects of fading. The selection of fade values by the controller is also performed with respect to a user-specified pattern analogous to that used for doppler shifts. In this case, pattern items may specify from 0 to 35 dB of attenuation and may contain 1,016 items.

2.8.2.1.4 Types of exciters

The three exciters of the SIG are labelled access 1, access 2, and jammer. Although very similar in structure, the various exciters perform different roles in the SIG, hence the difference in names. With respect to the exciter functions described above, the following distinctions are noted between the access exciters and the jammer exciter: (1) the access exciters will not perform PUL modulation; (2) the jammer exciter will not perform FM* and MFSK modulation, nor will it perform frequency hopping and fading.

2.8.2.2 RF Combiners

The outputs of the exciters are routed to one of four RF combiners in the SIG where they are translated to a new RF frequency, suitably attenuated, and then combined with one another and with other RF signals originating from external sources. The four combiners operate at UHF, L-band, X-band, and K-band, respectively.

A block diagram illustrating the combiner operation is shown in Figure 2.8.2.2-1. Of the five input ports indicated in the diagram, three are permanently associated with the three exciters and two are available for RF signals from external sources. The exciter inputs are independently translated to carrier frequencies appropriate to the given combiner and subsequently attenuated. Although neither the degree of frequency shift nor that of attenuation is dynamically variable, as is the case in the exciters, both of these parameters are user-selected to establish nominal signal frequencies and power levels.

2.8.2.2.1 UHF, L-band, and X-band combiners

The UHF, L-band, and X-band combiners are broken into two sections, uplink and downlink, which, with respect to the combiner operation indicated in Figure 2.8.2.2-1, operate independently. A more detailed block diagram of these combiners is shown in Figure 2.8.2.2-2. Note that both access exciter inputs are directed to the same section of the combiner, independently of the routing of the jammer exciter input. Note, moreover, that a pair of external signal input ports are provided for both uplink and downlink section. In each case, the signals pass through a switch whose status is user-selected.

Both uplink and downlink sections of the UHF and L-band combiners operate in the

* More precisely, there exists no internal source for frequency modulations of the jammer exciter; external sources may be used, however.

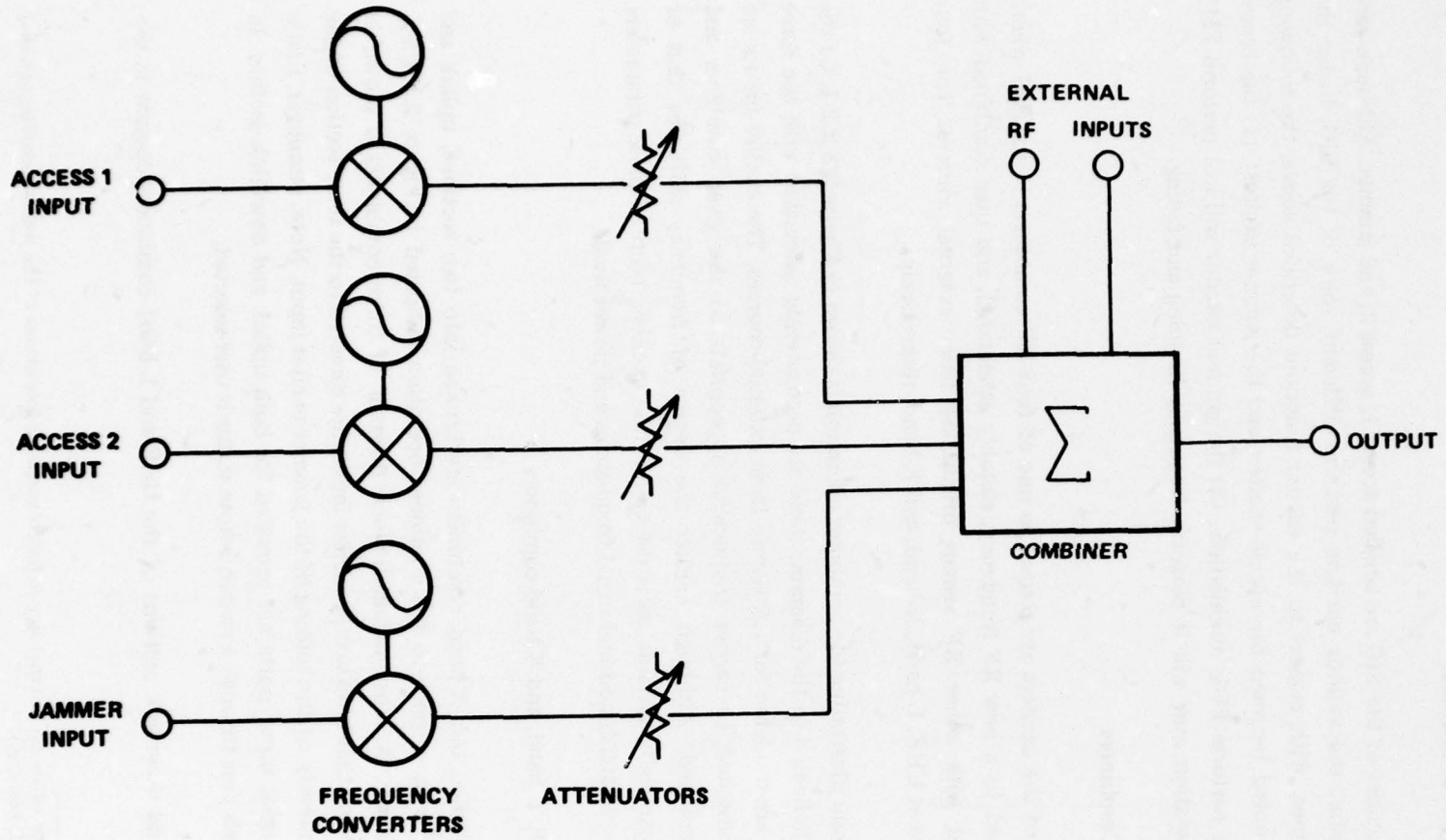


Figure 2.8.2.2-1. General block diagram of RF combiners.

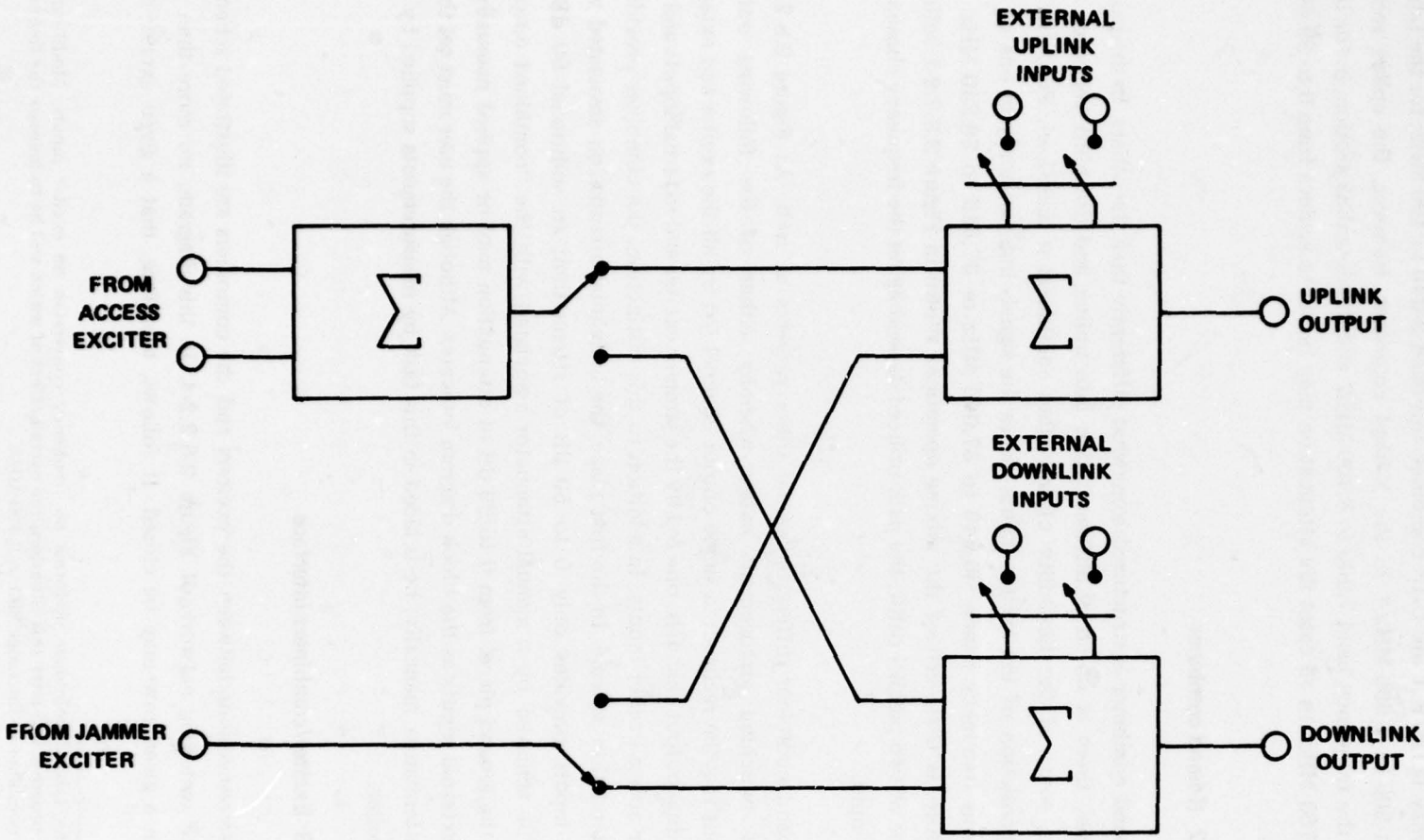


Figure 2.8.2.2-2. Detail of combining operation in UHF, L-band, and X-band combiners.

same frequency band. For the UHF combiner, this band is 240 to 400 MHz; for the L-band combiner, 1,200 to 1,600 MHz.* In the X-band combiner, however, the uplink section operates in the frequency band 7,900 to 8,400 MHZ and the downlink section, in the band 7,250 to 7,750 MHz. In all cases the attenuation may be set to achieve from 0 to 99 dB of attenuation.

2.8.2.2.2 K-band combiner

The K-band combiner is structured somewhat differently than the others. In the case of this combiner, there is no strict differentiation into uplink and downlink sections; as a consequence, some of the flexibility of the other combiners is sacrificed. Furthermore, frequency translation of the exciter inputs places the signals independently in one of two non-contiguous frequency bands—36,640 to 37,040 MHz or 37,840 to 38,240 MHz. Implied, therefore, is that each of the mixing operations shown in Figure 2.8.2.2-1 actually occurs in one of two parallel paths, the path utilized depending on the frequency chosen for that exciter input.

The K-band combiner differs in several other respects as well. As Figure 2.8.2.2-3 indicates, a switching arrangement exists whereby either of the following output configurations may be realized: a single output channel fed by all the exciter and external inputs; and dual output channels, one fed by the jammer exciter and external inputs and the other by the access exciter inputs. In addition to this distinction, the combiner provides a different attenuation scheme. In the first place, the individual attenuation associated with the exciter inputs provides only 0 to 50 dB of attenuation; an additional 60 dB of attenuation is achieved by a manual attenuator associated with the "combined output" channel. In the second place, from 0 to 99 dB of attenuation may be applied manually to one of the external inputs as the block diagram indicates. Although the user must set those two extra attenuators manually, he is aided in this task by measurements supplied by the digital controller.

2.8.2.2.3 Exciter/combiners interface

The interconnections between the exciters and the combiners are illustrated schematically by the switching network of Figure 2.8.2.2-4. In this diagram, no more than one crosspoint in a given row may be closed. It follows, therefore, that a given exciter may

* Currently the L-band combiner performs no frequency conversion on exciter inputs. Modifications currently in progress will alter this situation, the overall effect of which will be to increase the frequency band of this combiner to the range 800 to 2,400 MHz.

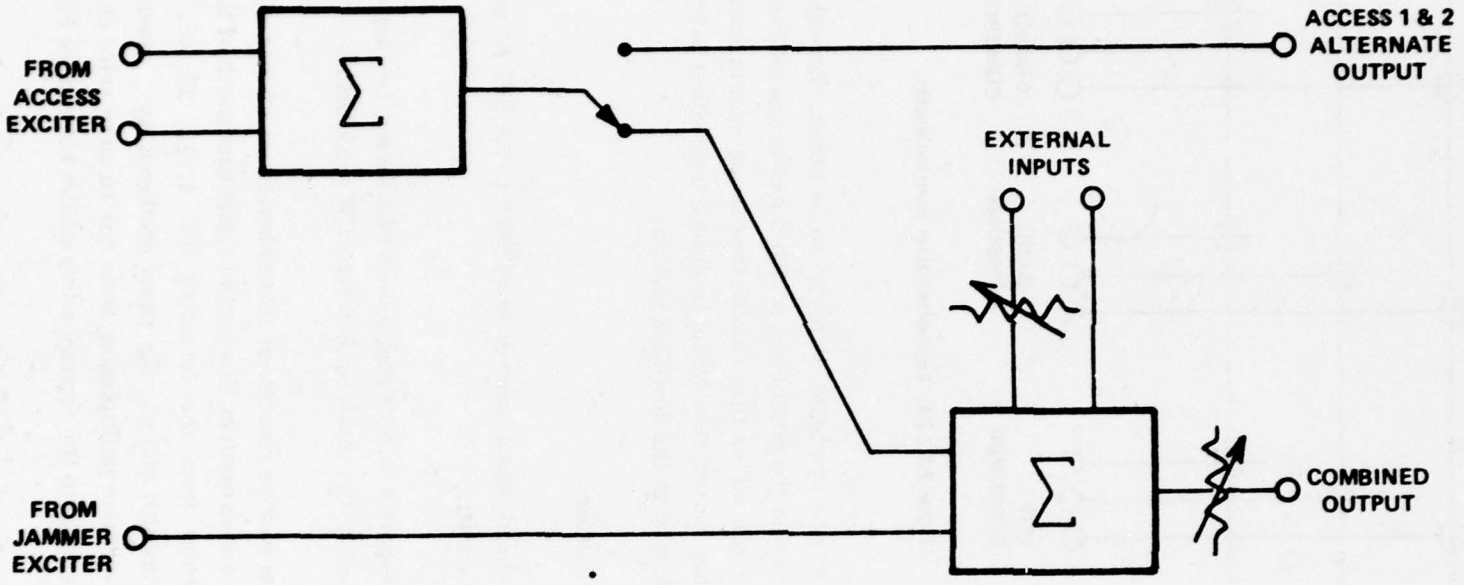


Figure 2.8.2.2-3. Detail of combining operation in K-band combiner.

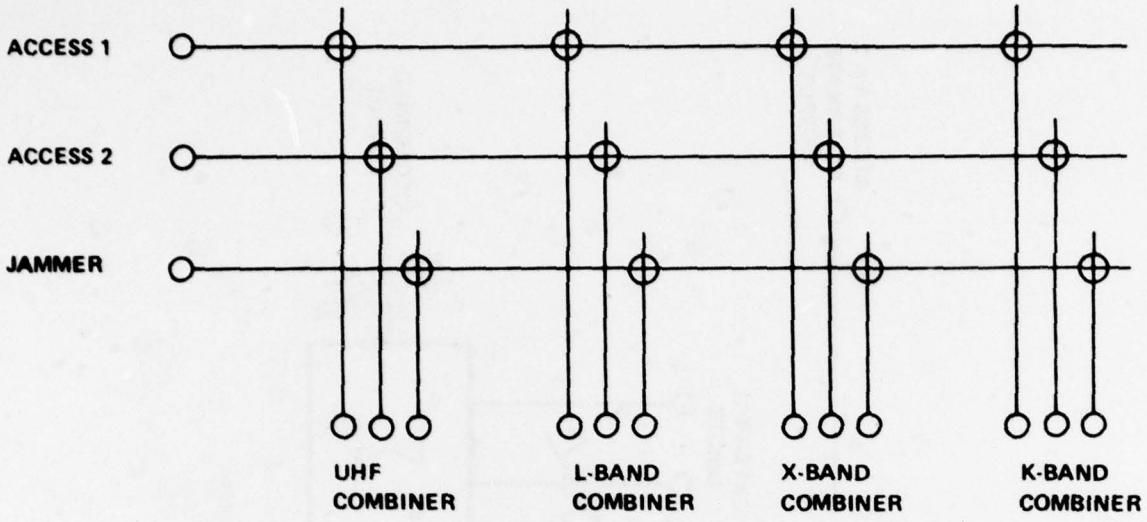


Figure 2.8.2.2-4. Exciter/combiner interrelationship.

be connected to at most one combiner. It is also to be noted, although the figure does not show it, that the switches in the combiners, which direct access exciter inputs to uplink or downlink sections, are ganged so that should two access exciter outputs be directed to different combiners, they nevertheless would both be routed either to the uplink sections of their respective combiners or to the downlink sections.

2.8.2.3 Baseband Sources

Five types of baseband signal sources exist within the SIG to achieve the types of modulation mentioned earlier.

The first type corresponds to no signal source whatsoever, i.e., a ground connection to the modulator of an exciter. The result is, therefore, CW modulation.

The second of these sources consists of two independent white noise generators paired respectively to the two access exciters. The output signal bandwidth of these generators may be selected independently from the following list: 4, 12, 20, 48, 144, or 240 kHz. Depending on the bandwidth chosen, the signal produced by a generator simulates that obtained by frequency-division multiplexing from one to sixty voice channels, each 4 kHz wide. This signal is then input to the corresponding exciter to produce FM modulation.

The third type of source consists of a pulse generator whose output drives an AM

modulator in the jammer exciter, producing PUL modulation. The pulse generator is capable of outputting a periodic on/off signal in which both the on-time and the off-time may be selected independently from the range 0.5 to 32,767.5 ms in increments of 0.5 ms.

The final source consists of the frequency synthesizer itself, which in addition to generating frequency hops and doppler shifts, may also be programmed to produce MFSK modulation. The modulating signal is again specified by a pattern as are frequency hops, the only difference being a frequency offset limitation of 10 kHz in the present case.

In addition to the internal sources just described, the SIG also makes provision for the connection of external baseband sources to affect either analog (FM) or digital (PSK, QSK) modulation in an exciter. The major consideration in qualifying such a source for use with the SIG is that it produces a signal compatible in bandwidth and power level with those produced by the internal sources.

2.8.2.4 Digital Controller

The function of supervising the elements described above is, as has been indicated, the task at the digital controller. This role is filled by a PDP 11/20 minicomputer with 44K of memory, extended arithmetic unit, and the following peripherals: dual cartridge disk drives, a magnetic tape unit, an ASR 33 teletype, two alphanumeric video displays, a card reader, a paper tape unit, a line printer, and an X/Y plotter. The teletype and video displays provide interactive interfaces with the system.

The computer is programmed to accept from the user sequence of commands which pertain to the various switches and parameters identified above. It interprets these commands and proceeds to configure, in so doing, checking for any equipment malfunctions. After performing the set-up task, the controller then supervises, in real-time, the operation of the SIG during an experimental run of the laboratory communication system, again monitoring equipment for malfunctions.

2.8.2.5 Illustrative Experimental Configurations

One illustration of the use of the SIG is provided by the program for which CSEL was originally formed, namely an investigation of the possibility of using the Lincoln Experimental Satellites (LES) 8 and 9 with the Advanced Airborne Command Post. The satellites, operating at Ka-band (36-38 GHz), are identified with the transponder in Figure 2.8.1-2; they perform the functions characteristic of such a device, including decoding and

recoding. It follows, therefore, that the communication channel of interest consists of two separate links. The experimental setups described here actually use LES 8 or 9 to connect (see Figure 2.8.2.5-1) to connect these links; the next section describes simulating the satellite with the Programmable Data Terminal.

Figure 2.8.2.5-2 and 2.8.2.5-3 show CSEL configurations appropriate for testing air-to-ground and ground-to-air K-band communication channels which use LES 8/9, respectively. The K-band modem and Ka-band terminal shown in the figures represent qualification models identical to those proposed for actual use. The 3-ft antenna is controlled by the Ka-band terminals, the 10-ft antenna, by a PDP 11/45. All of this equipment is part of CSEL. (See Section 2.8.4.)

In Figure 2.8.2.5-2, showing the air-to-ground channel, the signal transmitted by the terminal is down-converted to 560 MHz and fed into one of the access exciters where the fade and doppler are introduced. The output of the exciter is routed to the K-band combiner where it is corrupted by a jamming signal generated by the jammer exciter. The resulting signal is then transmitted by the three-ft antenna to LES 8/9.

The signal returned from the satellite is picked up by the 10-ft antenna, preamplified, and relayed to the receiver of the Ka-band terminal, from which the final output is obtained.

Figure 2.8.2.5-3 corresponds to the reverse procedure of that just described. Note that

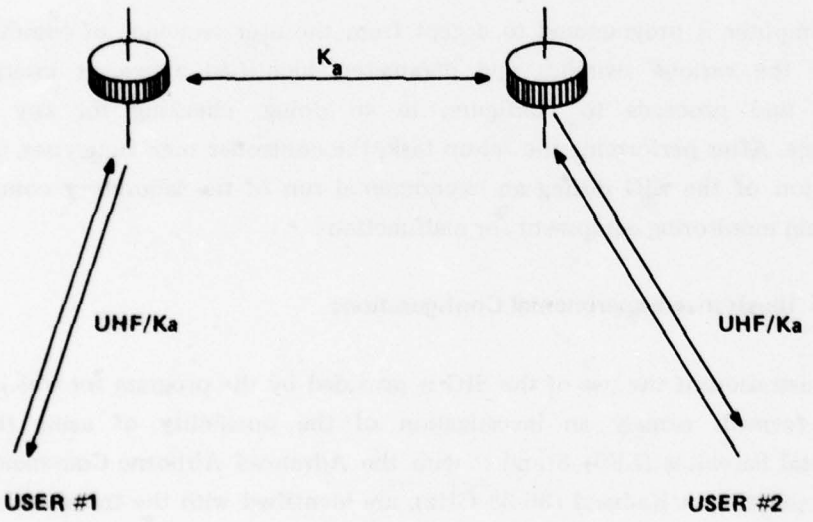


Figure 2.8.2.5-1. Typical LES 8/9 operation.

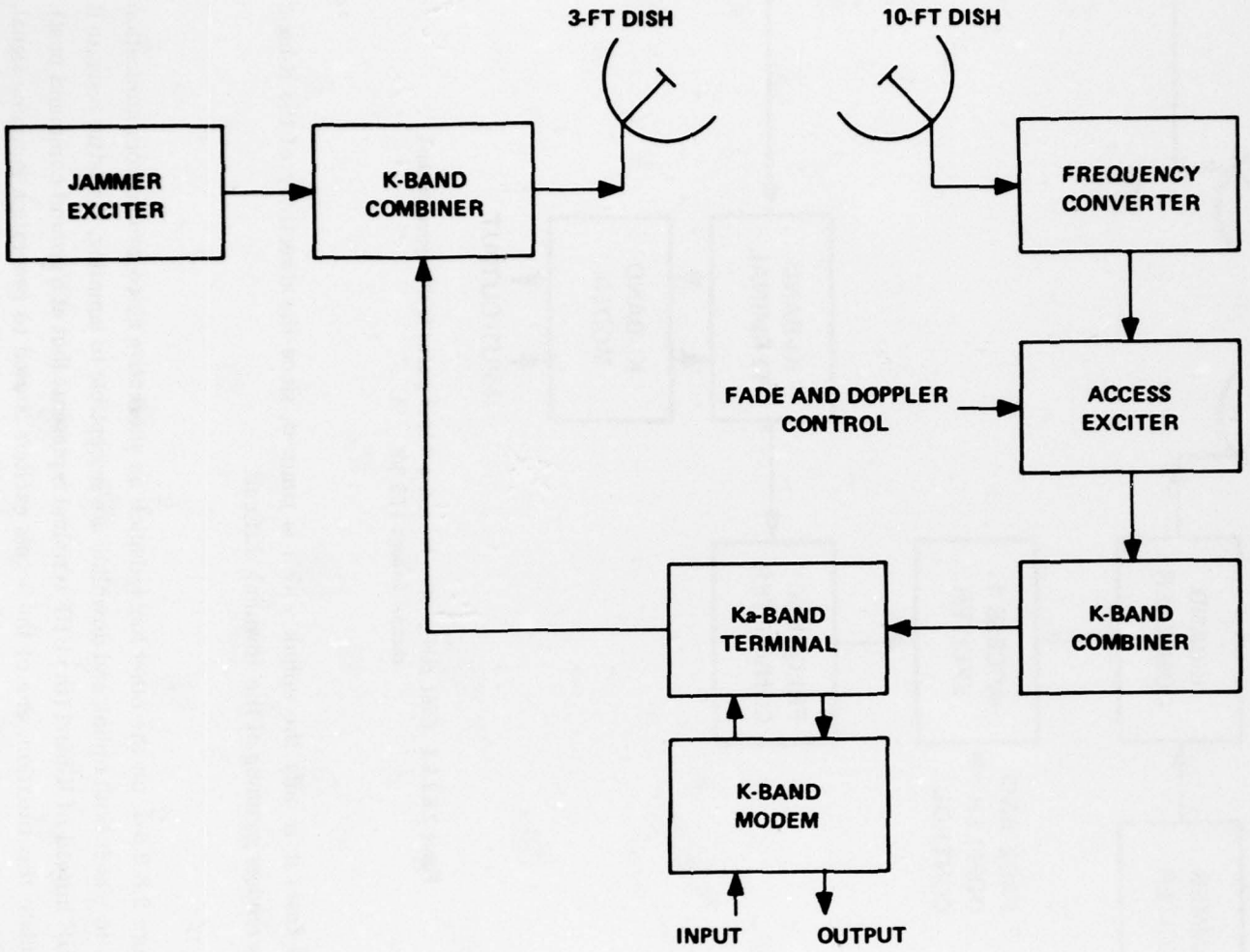


Figure 2.8.2.5-2. CSEL configuration for testing K-band air-to-ground communication channel through LES 8/9.

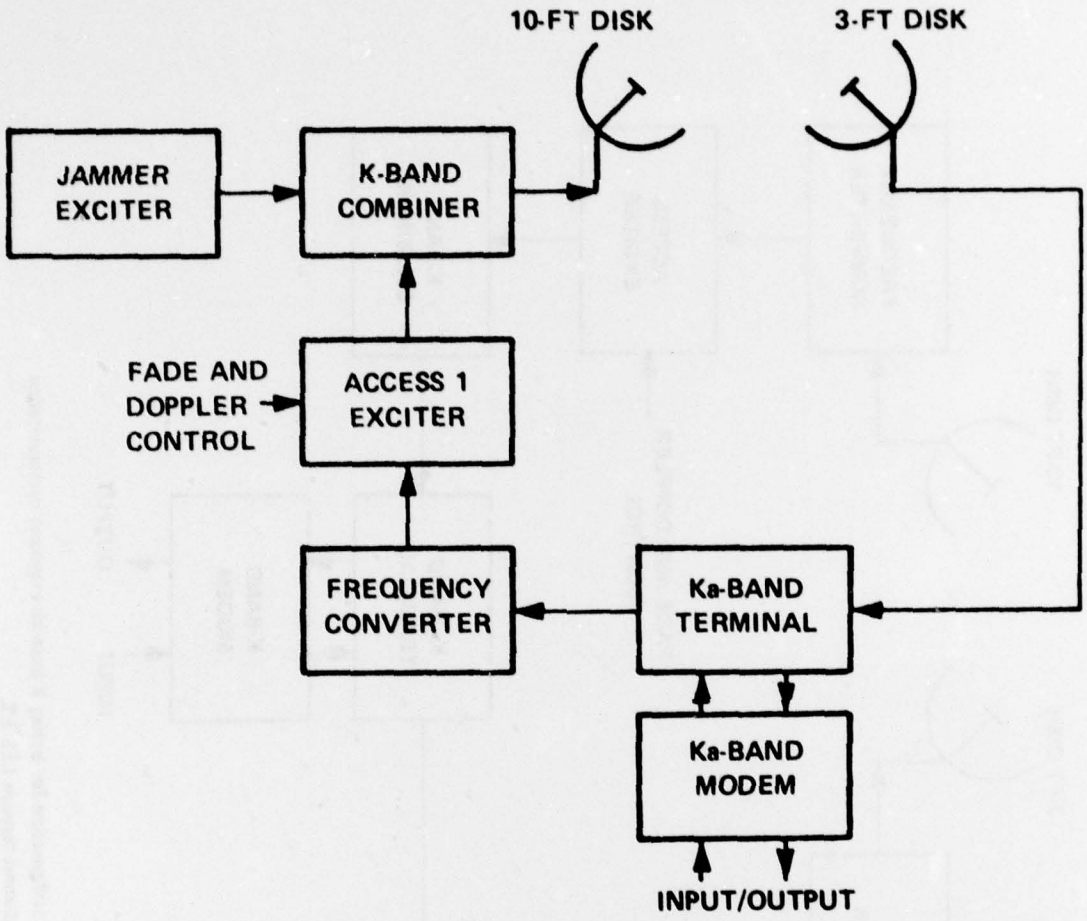


Figure 2.8.2.5-3. CSEL configuration for testing K-band ground-to-air communication channel through LES 8/9.

in both cases it is only the uplink which is jammed, since the directivity of the K-band antenna renders jamming of the downlink difficult.

Figure 2.8.2.5-4, on the other hand, depicts an alternative air-to-ground communication channel in which both uplink and downlink are susceptible to jamming, the latter because it is at UHF instead of K-band (the UHF terminal represents that at a ground command post). To simulate this situation, one of the access exciters is used to generate a jamming signal, while the other continues to provide simulated doppler shift and fading. In all other respects, however, the configuration shown is a direct analog of that in Figure 2.8.2.5-2.

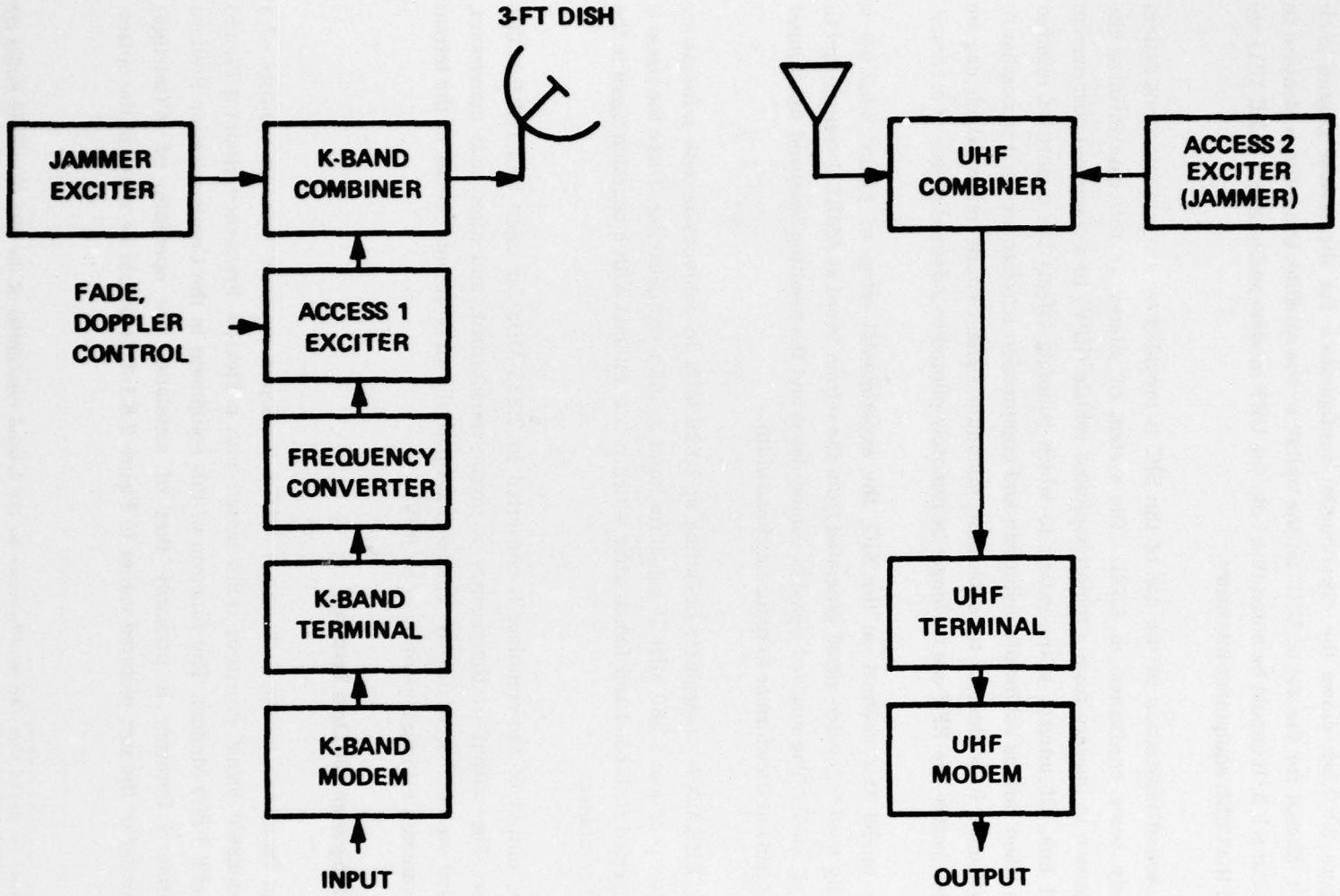


Figure 2.8.2.5-4. User configuration for testing K-band/UHF air-to-ground communication channel through LES 8/9.

Figure 2.8.2.5-5 shows the experimental configuration for the reverse ground-to-air channel. Except for the use of UHF on the uplink to the satellite, this setup is identical to Figure 2.8.2.5-3. It should be noted that both the UHF modem and terminal (ARC 151) are part of the CSEL equipment inventory.

A second illustration of the use of the SIG is provided by a communications system currently being configured in CSEL. The system of interest in this case involves the transmission at imagery from a remotely-piloted vehicle (RPV) to a command post over an L-band link. Of interest is the extent to which jamming affects the quality of received images when various bandwidth reduction and compression schemes are used to combat it. In particular, it is desired to determine minimum signal-to-noise ratios which can be tolerated before the RPV can no longer be remotely piloted to a desired degree of accuracy.

To model this channel on the SIG, the experimental setup of Figure 2.8.2.5-6 is currently used.* A video signal generated from the terrain board at AMRL is transmitted to CSEL at AFAL. The received signal is demodulated and the resulting baseband signal input to a Hadamand transformer to reduce its bandwidth.

A TERRACOM transmitter operating at 1,825 MHz in conjunction with a frequency converter produces a 560 MHz IF signal for input to the access 1 exciter. There the signal is up-converted to L-band and faded, after which it is combined with a jamming signal in the L-band combiner.

The output of the combiner is converted to 2,225 MHz for input to a TERRACOM receiver. The output of the receiver is inverse-transformed, and after scan conversion, displayed on a CONRAC display. The feedback path shown represents control of the terrain board camera to simulate piloting of an RPV.

2.8.3 Programmable Data Terminal

The Programmable Data Terminal (PDT) developed by GTE Sylvania consists of a Programmable Signal Processor (PSP) mated with a Flexible Frequency Hopping/Pseudo Noise (FFH/PN) Modem. The function of this equipment in the Communication Systems Evaluation Laboratory is primarily that of simulating the operation of a (satellite) transponder of the sort pictured earlier in Figure 2.8.1-3. It should be noted at the outset,

* It should be noted that the modifications to the L-band capabilities of the SIG mentioned earlier are motivated by this experiment.

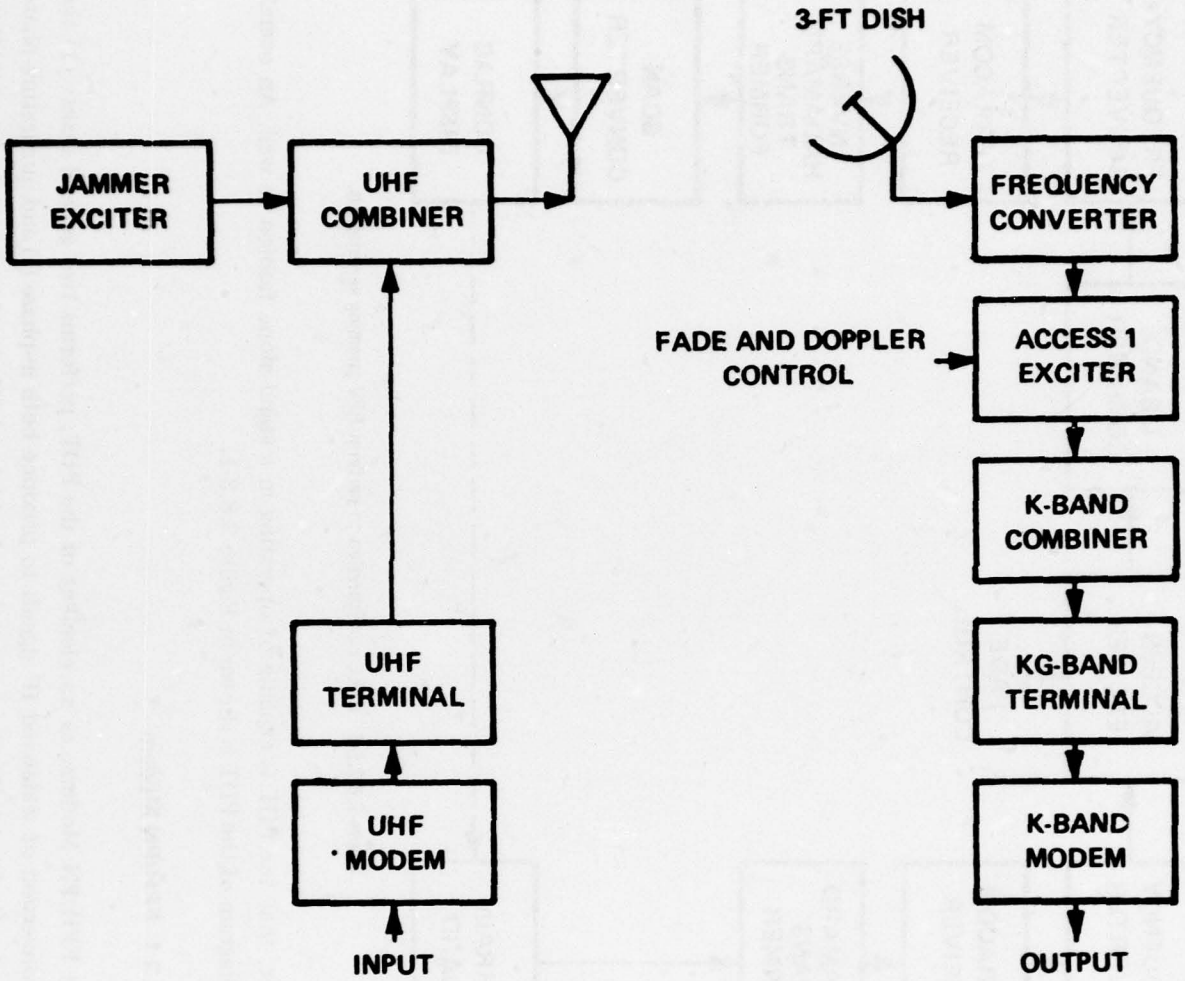


Figure 2.8.2.5-5. CSEL configuration for testing UHF/K-band ground-to-air communication channel through LES 8/9.

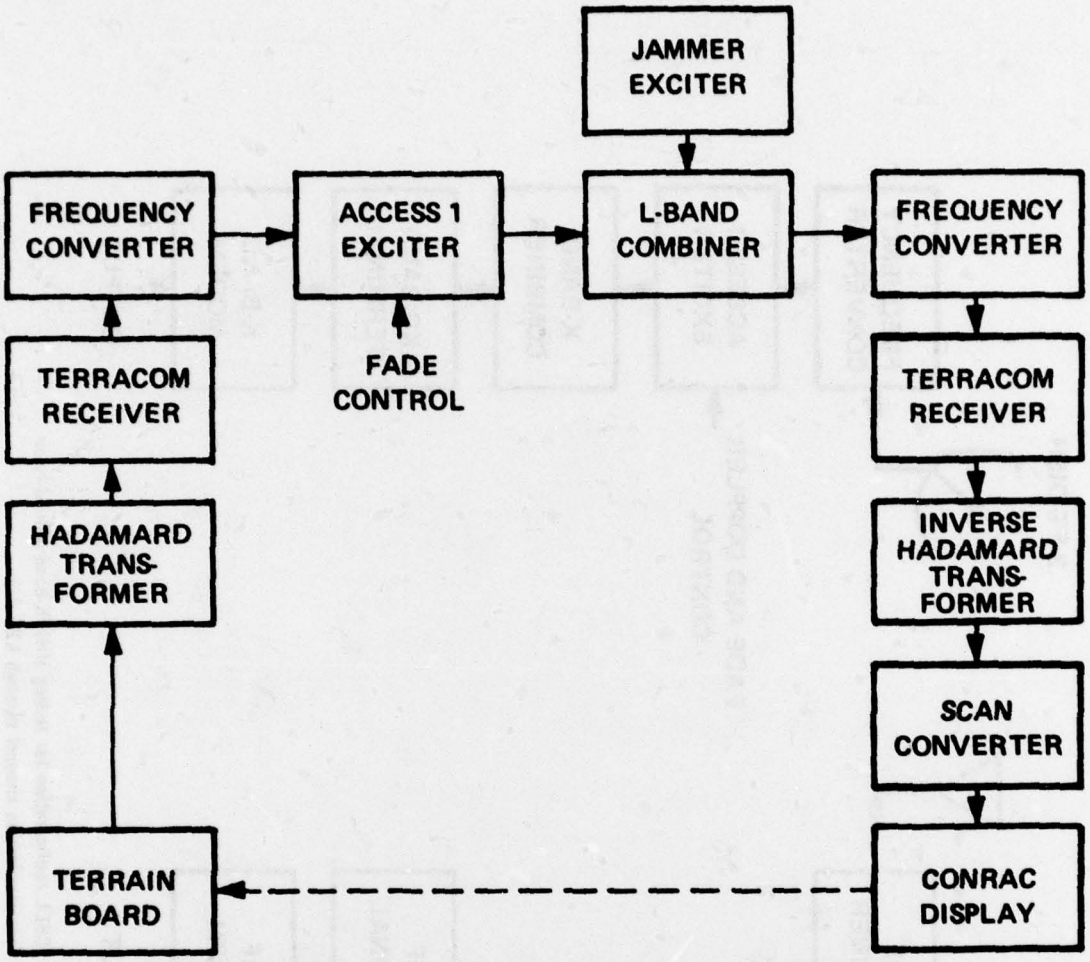


Figure 2.8.2.5-6. CSEL configuration to perform RPV jamming experiments.

however, that the PDT is capable of operating in a stand-alone fashion as well. An overall block diagram of the PDT is shown in Figure 2.8.3-1.

2.8.3.1 FFH/PN Modem

The FFH/PN Modem, as an element of the PDT, performs two general tasks: (1) the down-conversion of wideband IF signals to produce both in-phase (I) and quadrature-phase (Q) baseband signals; (2) the generation of wideband IF signals which are either FSK or QSK modulation (or both). In both cases the nominal IF carrier frequency used is either 70 or 700 MHz. Thus the modem may, in principle, be interfaced directly to either a VHF or UHF transceiver terminal. Likewise, the modem can operate in a wraparound fashion as well.

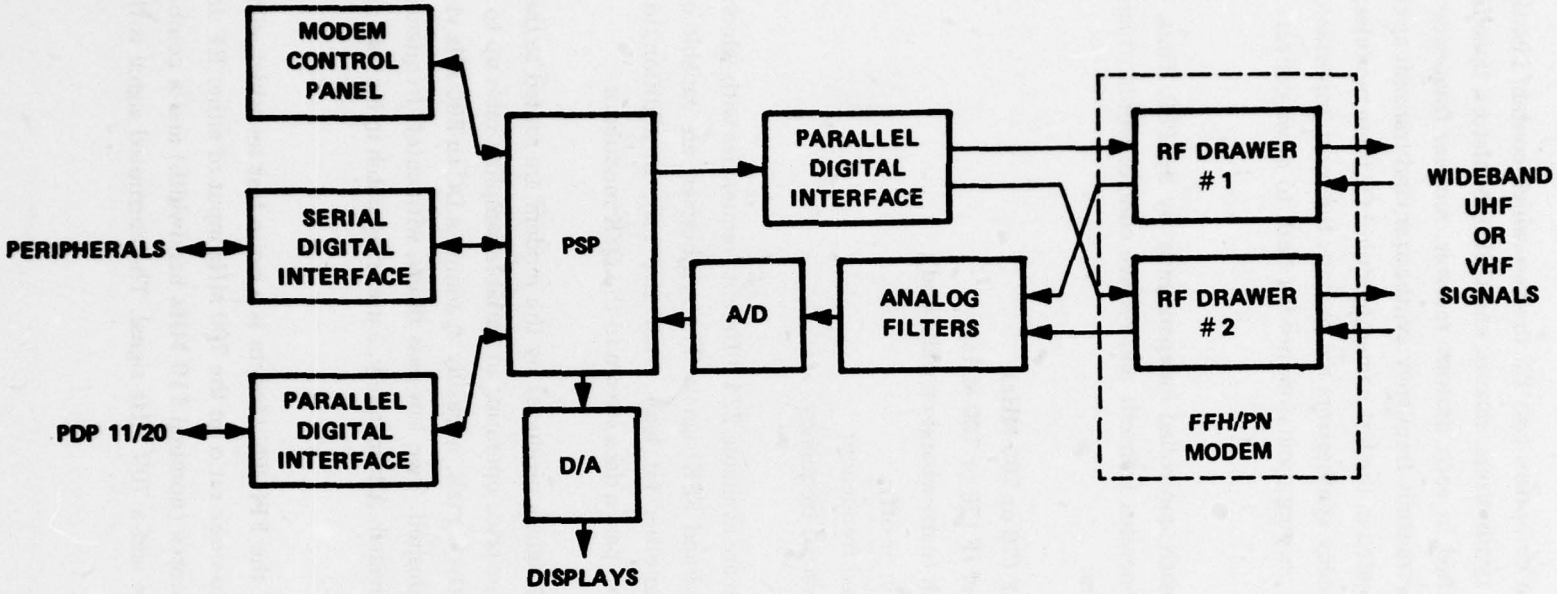


Figure 2.8.3-1. Programmable data terminal block diagram.

NOTE: Two primary components are the Programmable Signal Processor (PSP) and the Flexible Frequency Hopping (Pseudo Noise (FFH/PN) Modem.

Physically, the modem comprises two RF drawers which contain identical components in the form of frequency synthesizers, mixers, and QSK modulators. Specifically, five functional modules are identified in each drawer: receiver, receiver frequency synthesizer, receiver auxiliary generator, transmit frequency synthesizer, and transmit auxiliary generator. (The QSK modulator is located in the last named module.) Of these modules, it is only to be noted here that the frequency synthesizers are perhaps the most important, since it is their function to act as rapidly varying oscillators needed both to down-convert and to generate frequency-hopped signals.

The RF drawers are each controlled independently by the PSP. Thus, with respect to either drawer, the PSP generates transmit and receive control words which determine the following model parameters:

1. Nominal receiver IF (70 or 700 MHz);
2. Nominal transmitter IF (70 or 700 MHz);
3. Receiver bandwidth (narrowband or wideband);
4. Transmitter key (on or off);
5. Receiver synthesized frequency;
6. Transmitter synthesized frequency.

In the last two items, there are available 224 different frequencies with which to process and generate frequency-hopped and FSK signals. The synthesizers are capable of rapid shifting from one frequency to another (at least every five ms). In addition to the items just mentioned, the PSP can also pass a data stream to the QSK modulator.

The I and Q baseband signals produced by the modem are routed to the PSP through a commercial 12-bit A/D converter operating at variable sampling rates up to 50 kHz. Before being digitized, however, the signals, normally spanning a DC to 600 kHz video bandwidth, may be analog filtered if desired. Two low-pass filters, with cutoff frequencies of 1200 and 3500 Hz are currently provided. Alternatively, a user may patch in his own filter to process, for example, FDM signals.

A final component of the FFH/PN Modem is a noise test set which allows the user to create a controlled signal-to-noise ratio on the 700 MHz input of either RF drawer. Thus the test set contains a noise source (nominal 110 MHz bandwidth) and a combiner to sum the output of the noise source and a 700 MHz signal. The corrupted signal is then sent to the chosen IF receive port.

2.8.3.2 Programmable Signal Processor

The PSP is representative of current digital computers of the same sort in its ability to perform in real time a variety of high speed digital signal processing functions: detection, filtering, coding/decoding, etc. Thus, with respect to the FFH/PN Modem, the PSP is capable of handling simultaneously the following functions: (1) detection of the digital data carried on the I and Q baseband signals; (2) generation of output digital data to drive the transmit frequency synthesizer (FSK modulation) and/or the quadraphase modulator (QSK or PSK modulation); and (3) generation of frequency synthesizer control words to enable the reception and transmission of frequency-hopped signals.

To accommodate the throughput rate implied by these activities, the PSP has high speed memory (500 ns cycle time) and a high speed arithmetic unit (8 ms for a 256-point complex FFT). The machine uses 16-bit data words and 32-bit program words; currently 2,048 words of both data memory and program memory are available. A block diagram of the PSP is shown in Figure 2.8.3.2-1.

The PDP 11/20 (discussed above in conjunction with the SI6) acts as a host computer for the PSP. Thus, program development for the PSP can be carried out on the more convenient minicomputer and the results entered directly into the PSP. In addition, data can be passed between the two machines during real-time experimentation.

Complementing the digital I/O interfaces with the FFH/PN Modem and the PDP 11/20 are two serial interfaces for peripherals and three D/A outputs for analog display devices. Finally, a modem control panel is provided for monitoring and controlling the entire PDT operation through the PSP.

2.8.3.3 Application of the PDT

Use of the PDT to simulate the LES 8/9 satellites is straightforward. With respect to external hardware, one need only substitute frequency converters for the antennas shown in Figures 2.8.2.5-2 through 2.8.2.5-5, in order to convert between the RF frequency used (K-band or UHF) and 700 MHz. The PDT configuration appropriate to the application is shown in Figure 2.8.3.3-1, in which the operation to be performed by PSP software is displayed as well. These operations, of course, are those mentioned at the beginning of the preceding Section 2.8.3.2.

In addition to the LES 8/9 software, three other programs have been written to run on the PDT. With these programs, the system can operate as:

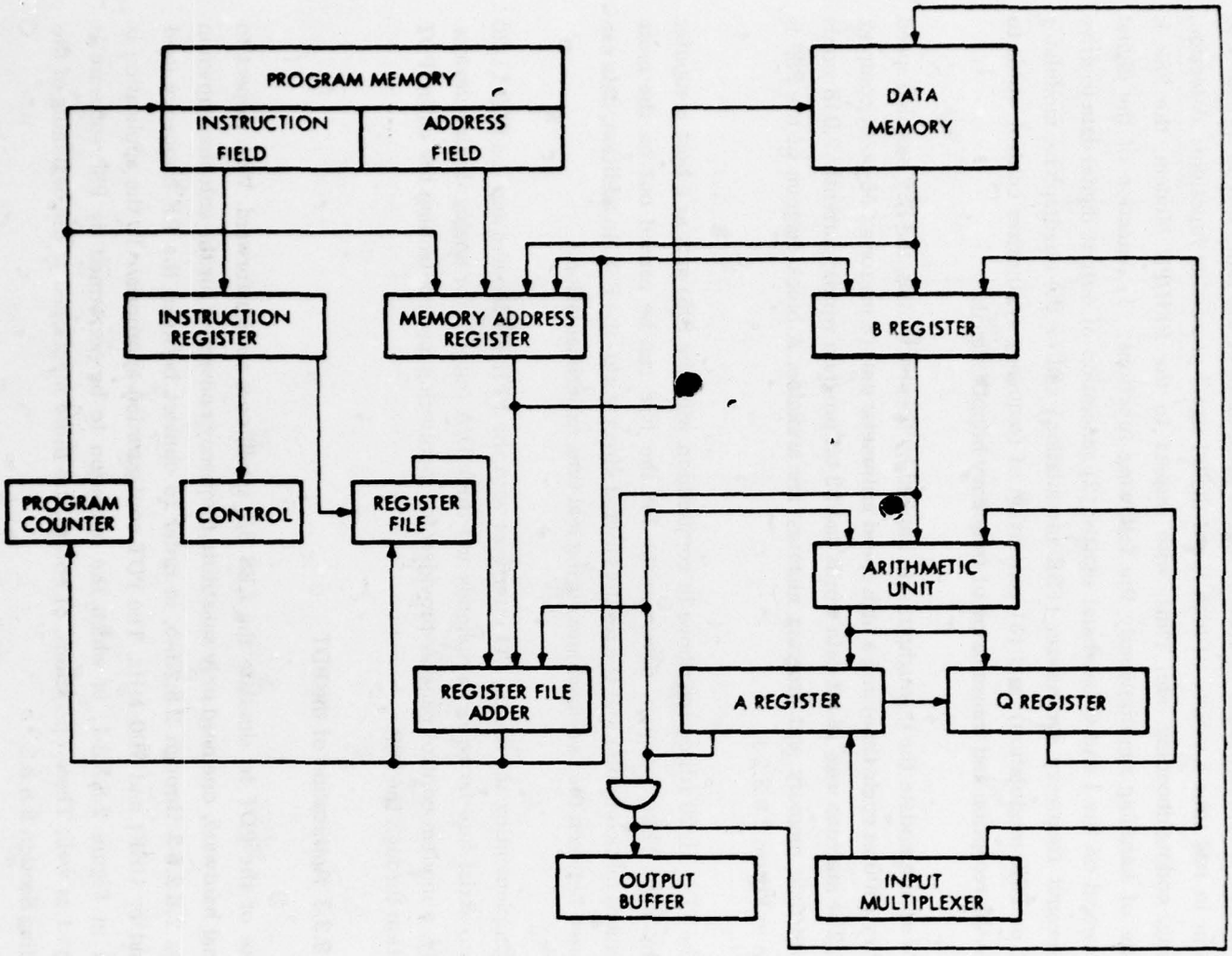


Figure 2.8.3.2-1. Programmable signal processor functional block diagram.

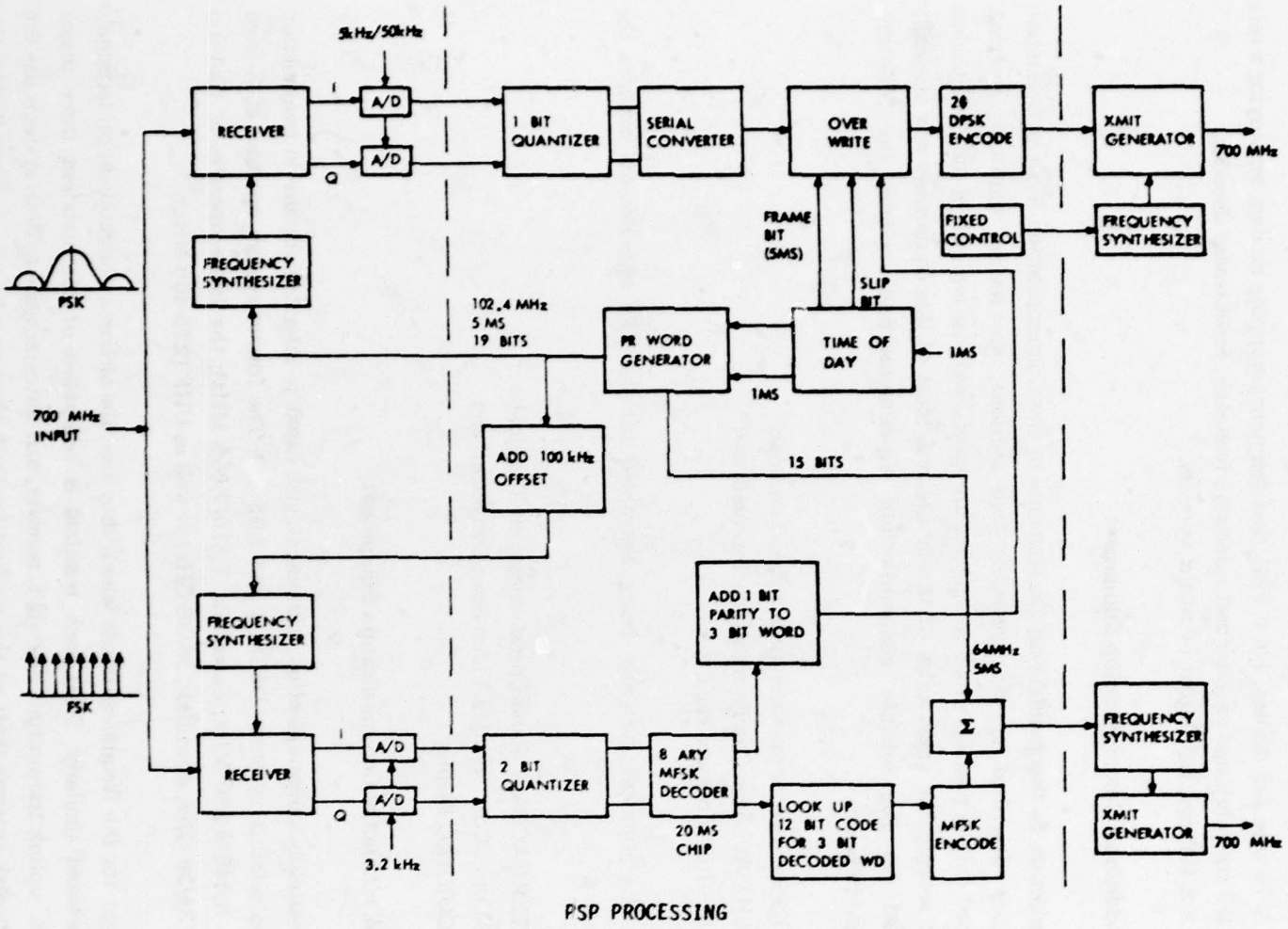


Figure 2.8.3.3-1. Programmable data terminal configuration simulating LES 8/9.

1. A 75 bps, full-duplex, 8'ary FSK, slow frequency-hopping modem employing a rate 1/3 convolutional encoder and a Viterbi decoder;
2. A 75 bps, full duplex, 8'ary FSK, fast frequency-hopping modem employing a rate 1/3 convolutional encoder and a decision feedback, coset leader decoder;
3. A 2,400 bps, full duplex cepstrum vocoder .

2.8.4 Additional Communication Equipment

In addition to the general-purpose elements of the Communication System Evaluation Laboratory described in the preceding two sections, are several hardware systems, mentioned only in passing, which to a degree are special-purpose equipment but nevertheless serve to solidify the capabilities of the laboratory. Most of this equipment was originally assembled for the satellite communication experiments and includes the following significant items:

1. Ka-Band Airborne Communications Terminal;
2. SHF Airborne Communications Terminal; and
3. Rooftop Antenna Facility.

Additional equipment currently being assembled for the RPV experiments includes the following:

1. TERRACOM L-Band Communications Terminals;
2. Motion-Compensated Video Scan Converter; and
3. CONRAC Display.

2.8.4.1 Satellite Communication Equipment

The satellite communication equipment just listed is intended for use in conjunction with two satellite systems, DSCS II and LES 8/9. The former system operates at X-band (uplink, 8,040/8,280 MHz; downlink, 7,315/7,555 MHz); the latter operates at Ka-band (uplink, 38/36 GHz; downlink, 36/38 GHz) as well as UHF (225-400 MHz).

Except for the frequency bands which they use, the airborne communication terminals are structured similarly. Thus each terminal is comprised of the standard three major elements: uplink transmitter, downlink receiver, and antenna system. Both systems use the same 3-ft disk antenna (part of the rooftop facility), the scan of which is controlled by the (tracking) receiver for spatial acquisition, and both provide automatic doppler compensation of transmitted and received signals.

The Ka-band terminal transmitter accepts IF inputs at 700 and 952.4 MHz. It upconverts either to the receive frequencies of DSCS II. The receiver produces IF outputs at 870, 700, and 70 MHz for LES 8/9 and at 700 MHz for DSCS II.

The SHF terminal transmitter accepts IF inputs at 70 or 700 MHz and upconverts them to DSCS II receive frequencies. The receiver performs the opposite function.

The major component of the rooftop antenna facility is a 10-ft parabolic reflector, designed to operate at K-band, with a cassegrain feed system utilizing an 11-in. hyperbolic subreflector. The antenna is mounted on a Scientific Atlanta pedestal allowing variable speed scanning through 360° of azimuth and 90° of elevation The entire unit is enclosed within a high transmittance, inflatable radome. Significant performance parameters of the system are shown in Table 2.8.4.1-1.

The antenna pointing system functions in one of four modes: designation, acquisition, active tracking, and passive tracking. The first mode, designation, refers to manual antenna pointing. It is followed by the acquisition mode in which the antenna is scanned automatically over a specified circular sector until the received signal exceeds a preset threshold, at which point the search is terminated. The scanning pattern consists of 13 concentric circles covering 2.5 to 25 planar degrees.

In the active tracking mode, a conical scan is used in which the main antenna beam is mutated at 65 Hz by rotating the subreflector. By setting the squint angle to yield a crossover loss of approximately 0.5 dB, the system can achieve a tracking accuracy of 0.02 degrees.

TABLE 2.8.4.1-1. PERFORMANCE PARAMETERS OF 10-FT K-BAND ANTENNA

Diameter	10 ft
Design Gain	59 dB
Sidelobes	-17 dB below peak
Polarization	Both right- and left-hand circular
VSWR	Less than 1.5
RF Isolation	20 dB between transmit and receive channels
Waveguide insertion loss	Less than 2 dB
Slow rate	±10 degree/s
Beam width	0.2 degree
Frequency band	Transmit: 34-40 GHz
	Receive: 34-40 GHz

In both the acquisition and active tracking modes control over the antenna is exercised by the Ka-band communications terminal described above. In the passive tracking mode, however, the antenna is controlled by a PDP 11/45 minicomputer; thus, using satellite refraction and ephemoris data, the computer derives appropriate azimuth elevation data and points the antenna accordingly.

2.8.4.2 Video Transmission and Display Equipment

Section 2.8.2.5 discussed the application of the SIG to the transmission of imagery from an RPV. In order to perform experiments in this area, CSEL is being equipped with selected video processing and display capabilities. In addition to the TERRACOM terminals, described earlier, which enable transmission and reception at L-band, the facility is to contain equipment required to study bandwidth reduction techniques (frame rate reduction) and bandwidth compression techniques (two-dimensional image transformation).

In the envisioned configuration, the terrain board camera (see Figure 2.8.2.5-6) is capable of reducing both the number of full video frames transmitted per second and the scanning rate used for each. As a result, a bandwidth reduction is achievable. At the display console, the reduced frame rate is adjusted by a PDP 11/05-based-motion-compensated scan converter. This device affects interpolation between successive received frames to smooth out the video display.

Between the two operations just described, each video frame is Hadamard-transformed and the resulting coefficients truncated to produce a bandwidth compression of perhaps 10:1. After transmitting this signal over the simulated channel, an inverse transform is applied to obtain the input to the scan converter. Both the direct and inverse Hadamard transforms are to be performed by special-purpose digital hardware now being constructed.

The equipment just described, when integrated with the SIG, will give CSEL a rather unique capability in studying the ability of bandwidth reduction and compression techniques to enhance the antijam performance of video links.

SECTION 2.8 BIBLIOGRAPHY

Anon., "Communication Systems Evaluation Laboratory," AFAL internal document, no date.

Anon., "Communication Systems Evaluation Laboratory," Computer Sciences Corp., no date.

Alwine, D. O., et al., "K-Band Terminal Simulator; Vol. I: Brief Description and Operating Procedure," AFAL-TR-74-81, Vol. I, Computer Sciences Corp., April 1974.

Alwine, D. O., et al., "K-Band Terminal Simulator; Vol. II: Hardware/Equipment Description," AFAL-TR-74-81, Vol. II Computer Sciences Corp., August 1974.

Alwine, D O., et al., "K-Band Terminal Simulator; Vol. II: Software System User's Guide," AFAL-TR-74-81, Vol. III, Computer Sciences Corp., May 1974.

Bradley, J. R., et al., "K-Band Terminal Simulator; Vol. IV: System Software Manual," AFAL-TR-74-81, Vol. IV, Computer Sciences Corp., August 1974.

DeLellis, J., et al., "The Programmable Data Terminal (FFH/PN Modem Modification)," AFAL-TR-74-328, GTE Sylvania, Inc., February 1, 1975.

Gaugler, S. P., "Data Link Simulation Programs," Contract No. F33615-75-6-1242, Systems Research Laboratories, Inc., October 1976.

*U.S.Government Printing Office: 1978 — 757-080/554