Order Number 1339071

A high speed network architecture for real time testing of an embedded computer system

Woelfer, Karl Alan, M.S.

The University of Arizona, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A HIGH SPEED NETWORK ARCHITECTURE FOR REAL TIME TESTING OF AN EMBEDDED COMPUTER SYSTEM

by

Karl Alan Woelfer

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 9

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Karl Alan Wolfer

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

J. W. Rozenblit — 12/11/89
Professor of Electrical Engineering — Date

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

TABLE OF CONTENTS
(continued)

LIST OF ILLUSTRATIONS

ABSTRACT

The Embedded Computer System Support Improvement Program, or ESIP, was begun by the U.S. Air Force in 1983 to find new cost effective ways of integrating, testing and maintaining the computers embedded in military airborne, spaceborne, and ground electronic systems. A major initiative of this program is the Extendable Integration Support Environment (EISE).

The EISE project involves design and development of a high speed network-based hardware and software integration and test environment. The ongoing work is being done jointly by U.S. Air Force civilian engineers at McClellan Air Force Base and TRW in Sacramento, California, in support of embedded avionics computers in the A-10 aircraft. The prototype design will be used to test and integrate various other aircraft and space systems.

The author was the EISE project lead system engineer from July 1986 through January 1988.

1. INTRODUCTION

System integration and testing present the engineer with numerous difficulties. Many of these difficulties are caused by limitations of the integration environment, while other problems arise due to the complexity and speed of operation of the system under test. The need for real-life testing situations is especially critical in computer systems controlling or navigating high speed aircraft.

Valid testing of highly integrated systems such as aircraft inertial navigation requires a test environment that will operate in real time, i.e. as fast or faster than the system under test. It must operate dynamically throughout the test, on a non-interference basis. The test environment must be "invisible" to the system under test, providing normal inputs that the system would receive in an operational environment. The test environment must at the same time be capable of monitoring the system under test and recording significant input/output signals for later analysis.

The objective of this work was to develop a standard network architecture along with simulation and test software for the A-10 avionics system that could be reconfigured for various embedded computer system testing

situations. This thesis describes the design of a high speed network of distributed processors and software which together provide a modular approach to real time, dynamic testing for the A-10 aircraft. (See Figure 1-1)

The work on this project is significant in that the EISE system is the prototype for new integration support facility design in the United States Air Force. The EISE project also incorporates the Ada programming language [1].

As project manager, the author was in charge of performing studies of the system requirements, reviewing research reports, determining the present and projected A-10 avionics system test requirements, and planning development. Later, in the role of lead system engineer, the author was head of the design team for the U.S. Air Force. He was responsible for deciding the A-10 Integration Support Facility (ISF) functional capabilities, and for defining system and subsystem requirements.

The author was actively involved in the system design process with a team of contract and government engineers. He worked on the system software design to produce the System Segment Specification in accordance with DOD-STD-2167, the Defense System Software Development Standard [2]. He proposed the standard network interface unit and selected a commercial VME bus system to be used for network communications processing. He also worked on the software

top-level design and timing considerations for the distributed real time executive and simulation data control software. The author worked with other software engineers to integrate the Ada programming language in the EISE design. He worked on EISE system designs to support other avionics and space systems, and co-wrote a support study for the Royal Australian Air Force's F-111 aircraft.

1983 - 1985

System Requirements
System Concept
Architecture Design
System Interface Specifications
Four-node prototype
Generic Aircraft Model software
Non-Real Time Network software

1986-1987

Standard Network Interface Unit
Real Time Network and Simulation Executive design
Operator Control Station
Database Interface
Hardware and Software Interfaces for the A-10 Avionics
Avionics Test Setup software

1988-Present

Ada Software rehost
EISE Design for the F-111 Aircraft
Diagnostic Software
Data Reduction
Expert System Applications

EISE Development Phases
Figure 1 - 1

2. SUPPORT OF EMBEDDED COMPUTER SYSTEMS (ECS)

2.1 ECS Definition

Over the past 15 years, technological advances have occurred in virtually all aspects of military defense system functions. This is particularly evident in sophisticated military systems that encompass an embedded computer system (ECS). An ECS is a computer or group of computers packaged as an essential and integral part of a system structure. The embedded computers themselves may be specially designed, for example, to satisfy military specifications, and have a specific role in the system. An example of an embedded computer in the commercial marketplace would be in the automotive electronics which monitor and adjust engine ignition and timing.

2.2 ECSs in Military Systems

Examples of military systems in the Air Force using embedded computers include aircraft, communication-electronics, electronic warfare systems, automatic test equipment, aircrew training devices, and aerospace ground equipment. A few of the emerging or state-of-the-art technologies being embodied in military systems include the latest microprocessors, Very High Speed Integrated Circuits (VHSIC), fiber optics, and Artificial Intelligence (AI).

2.3 Integration Support Facility (ISF)

An Integration Support Facility (ISF) is defined [3] as "an Applied Engineering Laboratory designed to support digital airborne or spaceborne systems and subsystem associated programs and equipments." There is in general a dedicated facility for each individual system supported. Avionics Integration Support Facility (AISF) is used as a generic term for most ISFs in the Air Force. Extendable Integration Support Facility is a relatively new term used to denote a single facility with application to a broad category of ECSs, including airborne, spaceborne, and ground electronic systems.

In general, the ISF has a twofold purpose; it provides the tools needed by software engineers to modify, test, and validate individual computer programs that reside in the embedded computers, and it provides system engineers the means for hardware/software integration and testing on the target system hardware. The ISF used at the Sacramento Air Logistics Center is composed of two separate, distinct systems for software development and integration. The extendable ISF combines both of these functions in one large networked system.

The reprogramming of embedded computer software is an engineering-intensive activity. Avionics Integration

Support Facilities have been used by the Air Force for ECS software support since 1975, when their use was pioneered at the Sacramento Air Logistics Center on the F-111 aircraft computers. The ISF utilizes a subset of the tools (computer resources and software) that were used by the military system's original developer. The ISF houses a technical staff of electronics engineers and computer scientists, and the host (development, simulation, and post-simulation processing) computers needed to provide the complete analysis, design, programming, testing, test data reduction, documentation, and configuration control of military system computers and their software. In addition, the ISF contains special purpose dynamic simulation subsystems to provide the real time environmental signals which flow to and from the actual defense system embedded computers.

Integration Support Facilities are developed specifically for the support of one particular military system. There are numerous hardwired interconnections peculiar to the needs of the system, making the support system difficult to modify. Often, multiple ISFs are built to support different models of the same military system.

The "support" of embedded computer systems includes the analysis and verification of hardware or software problems, development of solutions to these problems, and

finally test and validation of hardware modifications and software changes to the system. Following identification of a software problem or definition of a desired software enhancement, engineers will design new software and perform updates and recompilation of the embedded computer source code on a development computer. This updated executable software is then downloaded from the super-mini or mainframe development computer into the appropriate embedded computer, where it undergoes static and then dynamic testing.

2.4 ISF Components

There is much commonality between the equipment used and the functions performed by ISFs. All require some sort of software development computer system in addition to specific simulation software and specialized target hardware interconnections. Within classes of ECSs there are common simulation software functions that have to be performed. For example, all aircraft systems need an interface to supply environmental stimulus to their navigation computers. In addition there might be a radar system whose inputs will be simulated.

The following are hardware and software components of an Integration Support Facility in the Air Force:

The "Hot Bench" or "hot mockup" is employed to demonstrate functional performance of the ECS software on the actual test military system computers. The ECS hot mockup is a physical replica in the laboratory of the embedded computer system as it would appear in an aircraft, for example. In an aircraft ECS hot mockup, all inertial navigation system computers are interconnected over a data bus, and signals to and from the various computers can be monitored via extender cards. Specialized input/output and power panels may be added, but the essence of the hot bench is the integrated system of embedded computers. Certain

auxiliary equipment is generally required to provide real time simulation of the working military system computer environment to include generation of input signals, for example physical pressure and temperature inputs to an air data computer.

Hardware Simulation Kernels are custom software designed to simulate/stimulate inputs to the target system and process outputs to calculate future input stimuli. There are custom hardware interfaces and dedicated devices capable of simulating the digital and selected portions of the analog environment which surrounds the target system. The kernel may include a microprocessor and analog-to-digital conversion cards.

Computer Monitor and Control (CMAC) devices are used to observe and record target computer operation on a hot bench. A CMAC will trace the target computer's internal program execution on a non-interference basis, and can be used as a dynamic debugger to generate an interrupt signal based on execution of a certain instruction or access of a particular memory location.

"Static", or non-real time ground-based (laboratory) testing consists of measuring non-time dependent output responses to system inputs. It is used as a first step to validate new hardware in the system as well as

hardware/software compatibility.

"Dynamic", real time simulation systems completely test the ECS operational software and are used for total system hardware-software integration. The testing (simulation) environment must exercise the embedded computer system at its normal operating speeds. A military system is "flown" (in the case of our aircraft example) on the ground to verify proper operation before installing any changes in the actual aircraft for flight testing.

3. THE EXTENDABLE INTEGRATION SUPPORT ENVIRONMENT (EISE)

3.1 EISE Definition and Purpose

The Extendable Integration Support Environment (EISE) is the 1980s successor to the uniquely designed Integration Support Facility. While encompassing a target system hot mockup (the actual aircraft computer system integrated in the laboratory) as part of the prototype design, the EISE is primarily a real-time dynamic embedded computer system software development, test and integration facility. It is used to duplicate, in the lab, target system problems, to test software (and hardware) changes, and to verify final system integrity.

The Extendable Integration Support Environment expands the concept of ISFs to a generic test environment for multiple embedded computer systems. Its purpose is to provide the tools to modify, test, and integrate the Operational Flight Programs (OFPs) resident in the A-10 avionics computers or Line Replaceable Units (LRUs), and to facilitate the integration of new or modified LRUs onto the A-10 MIL-STD-1553 [4] Data Bus. These same tools are used to support the development and operation of the EISE system itself.

The need for a shift away from the present individual system-specific Embedded Computer System support facilities

to modular interconnected, integrated multiple function system support facilities was primarily economic. The total cost of the support computers and costs to develop the specific test software for each new system has become prohibitive, running into the millions of dollars.

The use of common building blocks for ECS support facilities reduces the proliferation of different terminals, operating systems, languages, and computer hardware within one facility. Operator efficiency is increased when the ISF staff is working with a standard support environment. Standards, when they are applied at the building block level, can provide the designer with a familiar media. The modular ISF building blocks allow the builder (system designer or test engineer) to readily expand an existing ISF or to more quickly create a unique design for a specific military system.

3.2 Requirements and Design Goals

The requirement for an A-10 aircraft computer Integration Support Facility was the genesis of the EISE project. A laboratory facility, more sophisticated and capable than the existing A-10 hot bench, was needed to test periodic system software and hardware modifications being made to the A-10 avionics. The new ISF also needed

to accommodate an extensive planned upgrade to the avionics computers, with similar integration and test capability for the new A-10 system configuration.

The A-10 hot bench was provided by the aircraft manufacturer as a means of allowing hardware technicians to test the system. The types of tests performed were limited to removal and replacement of computer boards or entire LRUs from an aircraft, testing them on the hot bench with known good avionics computers, and installation of hardware subsystem enhancements to test with the unmodified portion of the system. There was no similar capability for software modification testing.

Our approach to the design was to produce a combination aircraft simulator and avionics integration facility. The simulator would be geared to the test engineer (as opposed to being an aircraft pilot trainer), allowing inputs that are functionally the same as those from the pilot during a flight. The required integration and test features included data recording, data bus monitoring, selective aircraft computer emulation, interactive operator control of the simulation, color graphics displays, and post-test data analysis as well as software development capability for the test system and the system under test.

The EISE was conceived to provide a "generic" support

environment tailored to the A-10 aircraft, with extendability to virtually any embedded computer system. It consists of common hardware and software "modules" used as building blocks in a reconfigurable, high-speed Local Area Network (LAN). In this way the same physical hardware and software resources can be used to support a number of different embedded computer systems.

To summarize, the design goals of the EISE were as follows:

1) To support the test and integration of multiple embedded computer systems, and multiple types (Communication-Electronics, Electronic Warfare, Avionics) with a single facility,

2) To support interconnection of systems with dissimilar architectures, languages, and input/output requirements,

3) To allow extension and reconfiguration as the number of target systems to support is increased,

4) To support military system growth and component improvements, and

5) To incorporate currently owned ECS support assets (hardware and software) in the overall design.

3.3 Capabilities Overview

The EISE system architecture is designed to allow real-time, system-level dynamic testing of embedded computer system software. Our specific application was for the A-10 aircraft computers. The network design is flexible enough to allow reconfiguration for different embedded computer systems and testing situations.

With the exception of custom hardware interfaces to the particular embedded system computers, all hardware components are commercially produced items. The software in the system, on the other hand, consists of both commercially written software development tools and custom software for simulation and network functions.

The software simulation (applications) processing is distributed among multiple "nodes" on the network. Each node attaches to the network via a Network Interface Unit (NIU) which is a front-end processor performing data conversions to and from the network data format. This transparency of data conversion allows all types of machines on the network to communicate with each other in a common language. The EISE LAN may be interfaced to other EISE LANs via a gateway node that allows communication within the same building or across the country using the Defense Data Network or MILnet. Modularity in the system

allows for growth without altering the basic architecture, for example adding additional processors to perform more complex simulations.

The A-10 EISE provides a real-time simulation of the A-10 aircraft, together with significant environmental factors, to provide the total system environment for the A-10 avionics LRUs being tested. The A-10 aircraft software model is a six degree-of-freedom simulation, meaning that forces acting on the simulated aircraft are calculated for each of the x-y-z translational axes and the roll-pitch-yaw rotational axes. The EISE has a pilot station that is functionally similar to the actual A-10 cockpit. The EISE also has a software model of the A-10 Inertial Navigation Unit (INU) hardware and computer, enabling the operator to "fly" the simulated aircraft from this pilot station. Alternately, the simulation can fly in a preprogrammed, autopilot mode. It is possible to repeat missions so that test situations may be rerun, allowing the test engineer to change some parameters and to record different sets of data from one run to the next.

The EISE allows the test engineer to monitor and record selected data. This includes any ECS inputs and outputs, all MIL-STD-1553A data bus traffic, and any selected simulation parameters. All recorded data is tagged to identify its source and when it was recorded.

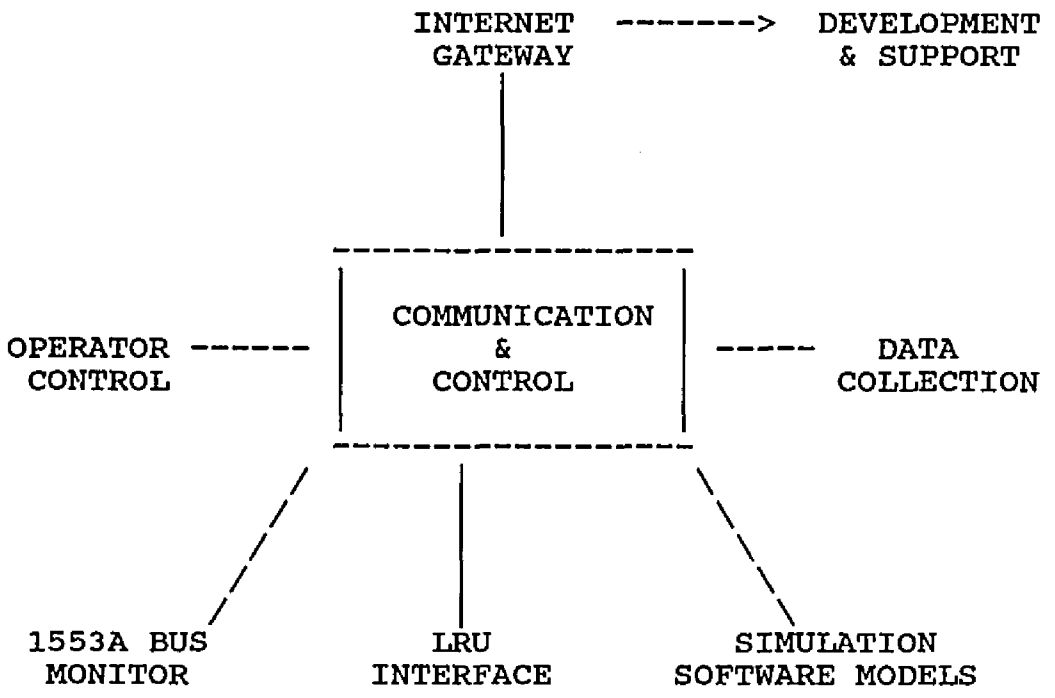
The EISE provides the capability to analyze any of the data recorded during a test. It is possible to correlate the different types of data using the time tag attached to the data.

EISE System Functions

The generalized EISE system functions (Figure 3-1) can be broken up into eight main categories:

a. The Communications and Control function provides communication among the other functions, and synchronization and control of real-time execution, requiring roughly 5% of a real-time simulation cycle to accomplish its tasks. It is made up of the Internet Protocol Suite of software (TCP/IP/FTP/TELNET), the Real Time Executive, Real Time Network Protocol and Simulation Data Control software, and the Proteon Network hardware. Non-Real Time functions are comprised of the same Internet Protocol Suite, User Datagram Protocol, and Network Executive software.

b. The Development and Support function provides the tools to facilitate target computer program development, modification, test and integration. It is made up of a host of software including compilers, assemblers, linkers, loaders, debuggers, text editors, file manipulation tools,



EISE System Functions
FIGURE 3-1

database, operating systems, and utilities libraries.

c. The Simulation Software Models is one of the most important system functions, creating the system environment for the A-10 avionics LRUs. They consist of an aircraft model, an environmental model, and an Inertial Navigation Unit model. These provide the input and output signals such that the rest of the computers in the system are unable to distinguish the model from a real source.

d. The Line Replaceable Unit Interface function allows communication between the LRUs and the rest of the computer systems in the EISE. It connects the avionics LRUs with the Communications and Control function.

e. The Data Collection system function collects and stores data during a real time test. Data gathered is time-tagged for the purpose of post-test data reduction and analysis. The EISE provides the capability of collecting sixty minutes of data at the rate of fifty parameters per 40 millisecond (msec) execution cycle, or a burst rate of five hundred parameters per cycle for a shorter period of time.

f. The Operator Control function provides the human interface to the EISE for control and monitoring of the real time simulation, both before and during the test.

g. The 1553A Bus Monitor function monitors and records the activity on the 1553A data bus during real time simulation. The bus monitor is capable of functioning at the maximum

message traffic speed of the 1553A data bus.

h. The Internet Gateway function is an interface between the EISE and the Defense Data Network, ETHERNET, and other networks including EISEs. The gateway is capable of accepting data from or transmitting data to another network which may have a similar or dissimilar protocol.

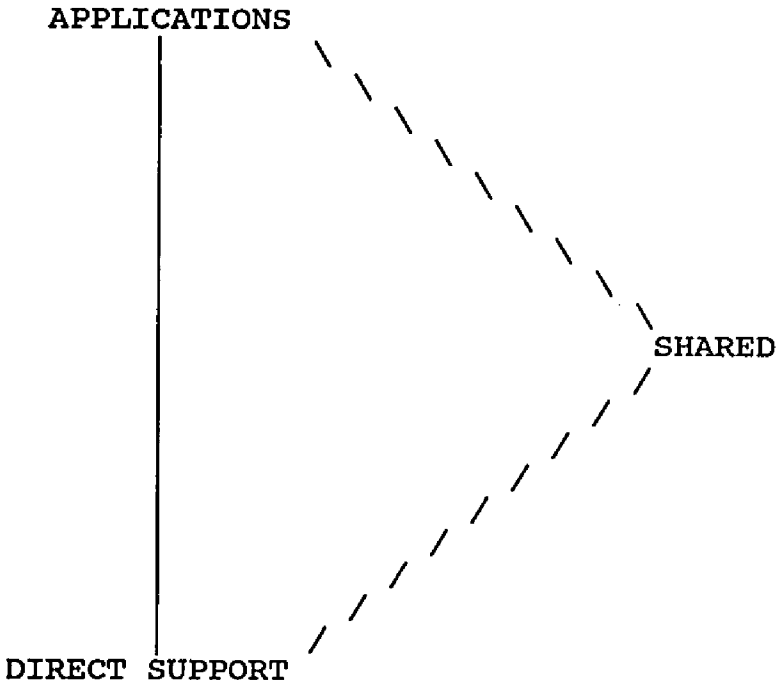
EISE Subsystems

The EISE can also be logically grouped into three major subsystems (Figure 3-2):

a. The Applications Subsystem and software includes both the computer work stations and the network communications and control devices, as well as software development and support tools, and an Ada Programming Support Environment. This subsystem is employed for EISE system host computer and network software development.

b. The Shared Subsystem and software includes the Data Base Management System, File servers, Internetwork Gateways, Data Storage Units, and Peripherals such as Printers, Tape Readers, and PROM systems.

c. The Direct Support Subsystem and software includes laboratory and military system resources needed to analyze, modify, and test the target computer (ECS) software. The complexity and capability can range from a stand-alone



EISE Subsystems
FIGURE 3-2

commercial development system to complete one-of-a-kind dynamic simulation systems with operator consoles and cockpit for interactive man/machine input and output.

Software tools permit the development of EISE system simulation software for applications and support. They also provide the capability to analyze proposed changes to ECS software, develop proposed changes to the ECS software baseline, test the changes prior to implementation in the baseline, analyze the results of testing, verify ECS performance after the baseline has changed, and test system avionics in a controlled environment.

The Hot Bench demonstrates functional performance of the ECS software in the actual military system computers. The hot bench is coupled with the EISE test system to provide real-time simulation of the avionics computer environment, including generation of input signals to the A-10 data bus. The EISE test system will perform post-test evaluation and interpretation of recorded output signals.

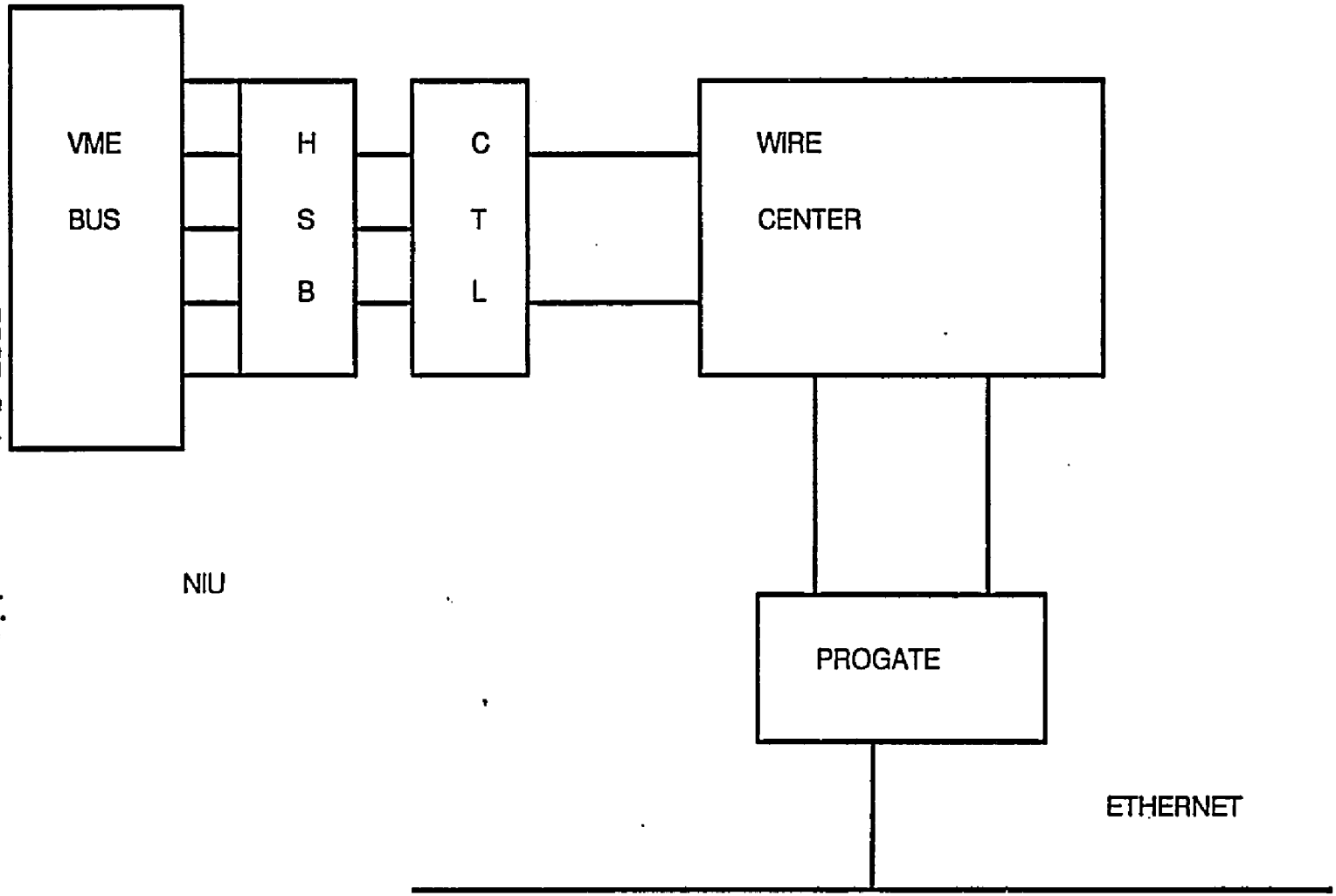
EISE System Description

The Primary Support computer for the EISE is a VAX 8650, a multi-user super minicomputer. It functions as the software development node of the EISE, and its hard disks provide mass storage capability for source software and executable and data files. Auxiliary support computers are

a Sun 280 workstation and a Symbolics 3675 development system for Artificial Intelligence software. All of these computers are peripherally connected over Ethernet [5]. They are interfaced to the EISE real time (ProNET) network through a Proteon network gateway. There is an individual Ethernet interface to each of the nodes.

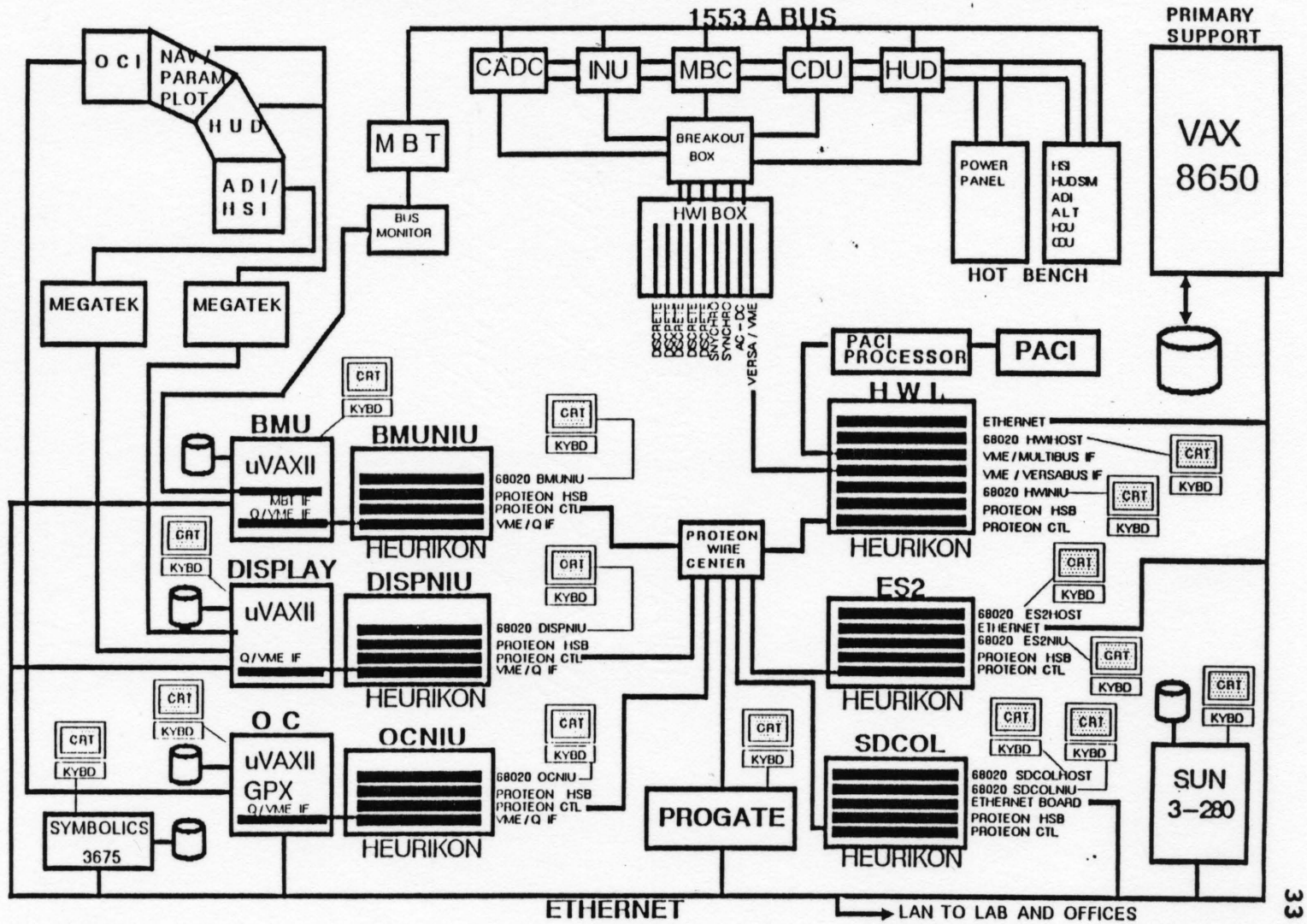
The interconnection mechanism for all of the following processors is the Wire Center of a Proteon ProNET network [6]. Co-axial cables, terminating at a standard Ring Control (CTL) board, branch out to all of the real time nodes in the system, and to the ProGate network gateway which connects to the EISE Ethernet network. The Proteon Host Specific Board (HSB), or CTL-to-VME bus interface that plugs into the card cage, and a single board 68020 computer function as the standard Network Interface Unit (Figure 3-3).

Each of the six nodes attached to the wire center is a Heurikon Motorola 68020 system. The basic Heurikon system consists of a VME card cage and a single 68020 processor board which is the NIU front-end. The processor on several of the nodes is a second 68020. In other nodes, VME bus interface cards connect to the Q bus (MicroVAX), Multibus (Intel), or Versabus (custom hardware interface cards) (Figure 3-4).



EISE Interconnection
FIGURE 3-3

A – 10 EISE PHYSICAL ARCHITECTURE



A-10 EISE Physical Architecture
FIGURE 3-4

The Environment Simulation processor was originally a Harris 1200 super minicomputer. The software was rehosted onto a single board 68020 computer. The Environment Simulation executes the equations of motion which are based on flight performance characteristics. These are comprised of: Stability and Control calculations; Engine, Earth, Atmosphere, and Terrain models; and Magnetic Variation calculations. These calculations and software models execute within a 40 millisecond cycle.

The Displays processor is a MicroVAX II computer. It is the driver for two Megatek graphics display computer systems which create the images for one color and two monochrome high resolution CRT screens at the Operator console. The operator can monitor the status of the simulation while the test is in progress.

The 1553A Bus Monitor processor is also a MicroVAX II which drives a Digital Technology Multiplex Bus Terminal (MBT). The MBT attaches directly to the hot bench A-10 aircraft data bus. It captures message data sent on the aircraft system bus.

The Operator Control Interface processor is a MicroVAX II GPX station. This is a full color, very high resolution display system with windowing capability to allow the operator to control and view multiple processes.

The Hardware Interface primary processor is a single board 68020. In conjunction with this, there is a Versabus card cage containing signal "conditioning" cards for discrete, synchro, and AC/DC signals. A breakout box allows access to internal aircraft computer signals via extender cards. Lastly, there is an Intel 80286 processor which gets aircraft control signals from the Pilot and Control Interface (PACI) throttle, joy-stick, and programmable switches.

Finally, the Simulation Data Collection function is hosted on a single board 68020 computer. Simulation Data Collection records selected data from the network for later analysis, and can capture any global simulation data broadcast as messages on the ring network.

Operating Modes

The non-real time mode is used to perform initialization of test parameters before a test, and analysis of collected test data following a test. This mode also provides support functions both for Operational Flight Program modification and integration and for development of the EISE system itself. In Test Definition the operator can set up simulation data files, while Quick Look and Data Analysis are for post-test review of simulation data. All support functions (use of compilers,

linkers, editors, debuggers, and other related software) are also non-real time.

The real time mode is used to run tests and to collect data for later analysis. Operator interaction is from the Operator Control Interface and optionally at the Pilot and Control Interface. The operator may start, abort, or stop the pre-defined simulation, control the flight of the aircraft or change the value of a parameter. Real time output consists of simulation status and simulated instrument displays to the operator monitoring the simulation, as well as files of recorded simulation parameter data for later analysis.

4. ARCHITECTURE CONSIDERATIONS AND DESIGN ALTERNATIVES

4.1 Centralized Host versus Distributed Processing

At the beginning of the system design stage, the expected amount of computer processing had to be estimated. For our specific application, the aircraft equations of motion and flight dynamics were the most processor-intensive activities. In the F-111 aircraft computer simulation stations in our avionics laboratory, this software is written in a mix of FORTRAN, JOVIAL, and Assembly languages and runs on dedicated (centralized host) super-minicomputers with a nominal three to five million instruction per second (MIPS) capacity.

There are a number reasons why we chose to use a distributed processing system for the EISE. First, this offers the maximum flexibility in the design and room for growth of the system to meet new requirements for additional processing or shorter (i.e. faster) simulation cycles. The degree of sophistication in the real time simulation would have to be limited in a single processor system. Secondly, the state of the art in high speed processing is moving toward distributed processing. A third motivation for a distributed design is that Integration Support Facilities traditionally have completely separated the development system from the target

system. This allows simulation testing to be run in parallel with software development and simulation data analysis. Our distributed design uses the same functional separation of the simulation testing and development. We use a super minicomputer system for software development, with additional microprocessors for the simulation testing functions, and all are connected over multiple networks.

4.2 Multiple Networks

Network design considerations were based on the two main types of tasks to be accomplished by the network, namely, the real time simulation communication and the non-real time (pre-simulation and post-simulation) support communication. The first category includes all simulation data that must be transported between nodes on the network each simulation cycle. The second category comprises all executable and data files to be downloaded to the simulation processors and the system under test (A-10 aircraft computers in our application), and the recorded simulation data which is transferred to a large host processor for post-test processing and data analysis.

The distribution of processing and allocation of tasks to nodes was an iterative procedure during the design and development. Once the processes to be performed and their

interfaces are defined, this enables seeing which data exchanges are necessary between the various node pairs. The simulation workload is partitioned according to individual node processing power so that no node is overloaded, and global simulation data is identified. Quantifying the internodal real time data exchanges will determine the minimum required network capacity for a given topology.

We decided early in the design that the EISE must be applicable to a wide range of microprocessors and minicomputers used as NIUs and processors. This influenced our network choice only in that we needed to be able to interface to many families of computers.

The high speed real time network design proceeded as follows. Each of the major real time System functions was allocated to a node on the network. Roughly, these node functions were 1) a hardware interface to the avionics computers, 2) a 1553 Data Bus interface for monitor and control, 3) a simulation data collection node, 4) an operator control/display node, 5) an aircraft environment simulation node, and 6) an Inertial Navigation Unit simulation node. The minimum number of nodes we needed was six for the distributed real time functions. Communications and Control was distributed in each of the nodes. One gateway node was included for interfacing to the

non-real time network and support processors. With the resulting seven node allocation, we wanted a modularly expandable network to be able to add nodes in the future.

We chose a 10 Megabits per second (Mbps) network by Proteon called the ProNET [6], which was later to become available in 80 Mbps speeds. The wire centers for this token-passing ring network are interconnectable, with a maximum of 256 nodes that can be on one network. We started with two, eight-node wire centers and a network gateway NIU that is part of the Proteon product line.

The Non-Real Time network chosen was the Ethernet [5], which runs at 10 Mbps. In the EISE, it supports "bursty" applications such as transmission of files to and from support processors where executable and data files reside in mass storage. The Ethernet is also expandable to support connection of a large number of nodes which can be physically more remote than the real time network nodes.

4.3 Network Interface Units (NIUs)

In network communication many of the tasks performed are relatively simple, but highly repetitive. These functions, which include data formatting, send, and receive, can make considerable demands on the time of the applications processing computer. It often proves more

economical to perform these functions in parallel in a separate computer. The main advantage is that the communications processing load removed from the main computer will increase the power available for applications computational purposes, while the hardware cost for this front-end microprocessor is minimal.

The term "node" used in this thesis refers to the combination of Network Interface Unit and distributed processor. The NIU can be thought of as a front end processor for the node. All conversions to and from a common network data format are performed here. The NIU also mechanizes the double buffering of global data, and synchronizes the update of local memory each cycle.

The wide availability and industry support of Motorola VME-bus products made this a natural standard to choose for our NIUs. There are interfaces from VME to all other standard microprocessor busses (Q, Multibus, Versabus). Each NIU was designed as a VME card cage with a 68020 single board computer performing the NIU functions. Attached to the VME bus is a custom network interface card (ProNET Host Specific Board) and a VME-to-distributed processor interface board. In the case of some nodes in the EISE system, the distributed processor is itself VME-compatible and simply plugs into the same VME bus. Families

of applications microprocessors and minicomputers interfacing to the NIU are VAX, Harris, Motorola, and Intel.

5. NETWORK TOPOLOGY CONSIDERATIONS

The networks in the EISE system serve as a bridge between the applications and development environments. They support the interconnection of a wide variety of non-homogeneous computing systems and systems built specifically for network operations. Additional network features are simple low-cost standard access methods, virtual connection of functions, high speed and relatively noise-free data transfer, and high reliability in a laboratory environment.

An upgrade which is being considered for the electrical twisted pair or co-ax real time network is a switch to optical fiber. The benefits of Fiber Optics are a wider bandwidth (data rate), and fiber's immunity from Electro Magnetic Interference (EMI), interception, crosstalk, and ground loops. Optical fiber is highly reliable with a very low error rate. The emerging Fiber Distributed Data Interface (FDDI) standard specifies speeds of 100 and 200 megabits per second. At the time of our design, commercially available networks provided speeds of 10 to 80 Mbps.

5.1 Performance Constraints for Real Time Testing

In order to meet the real time goals for our

particular application of the EISE, i.e. simulation testing and integration for the A-10 aircraft, a simulation cycle time of 40 milliseconds was required to match the bus data rate of the aircraft computer system. (The Inertial Navigation Unit actually runs at twice that speed). In the A-10 onboard computer system, data must be processed, sent, and received (for the next cycle) all during the 40 msec cycle.

Our choice for the real time network focused on Local Area Networks supporting low-level protocols. Their short transmission delays eliminated the need for complex buffers, flow control, and network congestion mechanisms. The Proteon network we chose performs the Physical and Data Link layer functions of the International Standards Organization (ISO) Open Systems Interconnect (OSI) communications model.

The speed of the network cannot eliminate the problem of two systems on the network communicating at different data rates. This problem is solved by standard front-end processors or NIUs that use higher level protocols to allow communication through replicated "shared" memory during the simulation. This is done by integrating portions of the network's file handling system within the individual NIU operating systems on the network.

An important factor in the timing equation is the

tradeoff between distributing the processing and the amount of data that must be shared between the distributed processes. On one extreme, all simulation processing could be done in a single machine. This would minimize the amount of data that needed to be sent on the network to display to the operator and provide inputs to the aircraft computers, but would at the same time give the single computer an unrealistic processing task. As the system simulation task is subdivided and functions are spread among multiple, loosely coupled computers on a network, the data traffic must increase.

The first analysis started with a budget for the percentage of the simulation cycle to be used for calculations by the applications programs, for sending and receiving simulation data over the network (a function of the network hardware), and for conversions to and from network data format. These are three serial processes that must be synchronized to assure data coherency. The application processes were allocated a maximum of 75% of the cycle time to run. This is followed by the last 25% of the cycle concerned with all of the front-end processing done in the NIU in software to append address headers, build messages, and to copy data to and from the network hardware Input Output (I/O) buffers, plus the electrical

transmission time.

It was liberally estimated that we would need to send 1500 bytes of data each cycle. The timing analysis for send and receive was done using a 10Mbps Proteon network for messages ranging in size from 100 to 2000 bytes, using both software timers and a logic analyzer. The individual portions of the total transmission time were broken down, and finally the performance of an 80Mbps network was extrapolated from the results.

The network performance test software started with a send routine for the various length messages. The routine set up the network hardware Command Status Register to start the send. Next the network hardware did a Direct Memory Access (DMA) operation. The DMA consists of controller software obtaining the commands and the address of the output data buffer, followed by loading the messages from host central processing unit (CPU) memory into the Proteon buffers for output. Once DMA is complete, the network hardware performs internal setup and begins to serialize the message onto the network. After the entire message has been sent, an Operation Complete interrupt is sent to the host CPU, which will reset the network hardware registers and clear the interrupt.

About 45 microseconds were required to setup any DMA operation, plus 215 nanoseconds (nsec) per byte actual DMA

time. Serialization in the network hardware takes 25 microseconds (usec) plus 810 nsec per byte. Interrupt time is a constant 3.5 usec. Interrupt Service Routine time is a constant 11.5 usec, send software 44 usec, and hardware 1 usec. Thus the total includes a constant 130 usec plus 1.025 usec/byte. The slowest transmission on a per byte basis was for a 100 byte message (2.32 usec per byte) and the fastest was for a 2000 byte message (1.09 usec per byte). These translate to an effective network speed of 3.44 and 7.34 megabits per second respectively.

Assuming that all times except serialization are the same for an 80Mbps network, the times become 1.62 and 0.381 usec per byte (4.95 to 21.0 megabits per second) for the 100 and 2000 byte messages, respectively. Depending on the choice of network then, the smallest messages could take two to three times as long to send, and it is for this worst case that we had to design. At the same time we realized that the 10MBps network would be inadequate.

Receive serialization from the ring network takes place in parallel with transmit serialization to the ring, and is therefore only charged once for the end-to-end timing calculations. The receive processing starts with an Interrupt Service Routine when data has arrived from the network. The receive software determines message placement

and builds control blocks for use by the DMA controller, and takes a constant 85usec. Network hardware internal operations and the interrupt processing together take a fixed 4usec. The DMA times are equivalent to those listed above for send, except that the fixed control block processing time is a bit longer due to more control blocks. Total receive time ranged from 167 usec for a 100 byte message (1.67usec per byte or 4.79Mbps throughput) to 584 usec for a 2000 byte message (.292usec per byte or 27.4Mbps throughput). The effective throughput for the 10Mbps network sending 2000 byte messages was measured at 9.95 Mbps.

There are two factors which affect the throughput from processor to processor. Node and ring speeds determine how fast packets can be placed in an NIU's random access memory (RAM) for output, and how fast the packets can be transported from sending to receiving node. Unaffected by ring speed is the internal time required for packets to be moved between the Pronet Host Specific Board RAM and the NIU RAM. For the 80Mbps network, serialization time is reduced to a point where messages can arrive from the network more quickly than they can be processed (copied to the NIU's RAM). This situation must be avoided through careful scheduling of messages in the system design.

5.2 Non-Real Time Network

The non-real time network is a concentric ring to the real-time network, providing an additional port to the distributed processors. The Ethernet 10Mbps standard was chosen. It provides the same Physical and Data Link layer support as does the token passing ProNET, and is coupled with File Transfer Protocol as well as the Transmission Control Protocol and Internet Protocol software to download executable and data files to processors on the network.

5.3 Hybrid Star / Ring Topology

The choice of topology depends on the application, the transportation medium, and the methods used to distribute data. The Ring and Bus have significant advantages in that they support multiprocessing and are easily expanded. A brief discussion of the various network topologies is in order. A Point to Point network interconnection scheme is simple, yet costly and the least versatile. Each additional node on the system must be connected to all existing nodes. In a maximally connected point-to-point network with N nodes, the number of interconnections is N * (N-1).

The Star network is simple, requiring only two lines for each node connection. Its main vulnerability is that a central node failure will disable the entire system.

The Ring network does not involve a central node. It is simple, requiring only one line to its neighboring node. In a bi-directional ring network, any single node failure still allows routing of network traffic to all other nodes in the system.

Both the Ring and Bus network topologies avoid the problems and inefficiencies associated with central processing. Centralized architectures, such as the Star, require every bit of information to pass through the central node. In a single CPU multitasking environment, only one task can have the CPU's attention at any one time.

The EISE is a multiprocessor environment where each computer runs autonomously at its own application, transferring information between distributed functions or processes only as necessary and keeping the network available for priority use. Adding or removing functions or processes does not significantly impact other functions on the network. There are no routing decisions to be made by a central node. The only routing requirements are that each node recognize those messages intended for it. Most simulation data is contained in replicated shared memory that is broadcast to all nodes.

The "star-shaped ring" network is a hybrid, combining features of both point-to-point and broadcast topologies.

The Proteon ProNET network is such a star-shaped ring (loop) topology, offering increased reliability and maintainability, and therefore more fault tolerance than either the star or ring schemes can singularly offer.

To explain the network in the context of a Star network, we consider first of all that the network is logically a "loop", with each node or Network Interface Unit receiving a circulating message in sequence before passing it on to the neighboring node. However, lines to and from each node pass through a relay in a central "wire center". Each relay is energized as long as there is a cable connection to its node. These relays in the wire center are capable of switching failed nodes (due to cable breakage, power failure, or software problems) out of the network so as not to break the loop, and thus from a fault tolerance perspective the topology is also a "star".

While each NIU is addressable to allow it to receive unique messages, a group of nodes may fall under the same broadcast address. This allows larger packets (messages) of common data to be sent which, as was pointed out earlier, increases the transmission efficiency.

5.4 Protocols - Message Circulation Schemes

Protocols are a set of interface specifications and

part of the control structure of the data transmission. They are layered to provide a hierarchy of tasks that need to be performed in formatting data from the neighboring layers. A protocol will specify data packet headers with fixed fields, in fixed locations, and addresses which translate directly to queues, buffers, ports, or processes without the need to consult lookup tables.

A deterministic control strategy that supports the real time needs of an ISF is found in the token ring. Control is passed sequentially around the ring from node to node, giving equal access. The worst case wait time for a node on the ring to be able to transmit can be calculated assuming all nodes transmitting the maximum length message. If N is the number of nodes, this time is the product of (N-1) and the transmission time for the longest message.

The message circulation scheme in the real time network is a hybrid between point-to-point and broadcast addressing. There is a single circulating token (the bus arbitration unit for this centralized channel allocation method) that is seized by a node to gain sending access to the network. A message, addressed to one or more nodes, is shipped out of one node and the token appended to the end (there may have been other messages on the network or just a token). The network hardware at each node stores only 10

bits at any time as the message is circulating, and if it is addressed for that node or has a broadcast address, the message is copied into the node's input buffer. This might be considered a "limited" store-and-forward. The message will be "drained" from the network by the sender unless it was refused at the destination (indicated by status bits in the header) in which case it is retransmitted.

The EISE real time simulation testing requires a virtual circuit to ensure processes can be serviced on a predetermined schedule or priority. The token passing scheme gives a deterministic response time for access to the network, which is very important for real time applications. A recovery scheme is built in to the network and will recreate a lost token after a predetermined time.

The control mechanism for the non-real time Ethernet communications channel is Carrier Sense Multiple Access with Collision Detect (CSMA/CD). CSMA/CD performs a recovery scheme when a collision results from two nodes attempting to enter messages (file transfers for example) onto the network at the same time.

6. HARDWARE REQUIREMENTS AND COMPONENTS OF THE EISE

6.1 The Software Development Environment

The EISE system architecture is designed for software development for both the (target) embedded computers and the EISE (host computer) system itself. The Operational Flight Programs which run the ECSs are modified on a host computer, then cross-compiled into the native language of the specific target processor (aircraft embedded computer). In the same way, software to run the EISE simulation system is upgraded or modified to provide additional or improved functionality for the simulation environment.

The Primary Support node is a large, multi-user super-minicomputer (VAX 8650) where most of the software development and documentation is performed. This is the non-real time portion of the system and is not directly involved in the simulation process. It is the repository for all executable simulation code and application software for each of the NIUs and host processors. The Primary Support node is peripherally connected over an Ethernet link through the Internet Gateway (Progate) to the real time EISE network.

6.2 Download Capability

Downloading, or transfer of executable and data files,

is performed between the Development and Support nodes on the non-real time network and the embedded computers and distributed applications processors on the real time network. The Internet Gateway is a Proteon product that enables connection of the EISE network to the Defense Data Network [7] or MILNET, other long haul networks, or Local Area Networks at the same facility. Downloading is a non-real time feature that incorporates the Transmission Control Protocol and Internet Protocol (TCP/IP) for transfer of data files and permits sharing simulation programs and software tools between remote users.

An Ethernet link to the Internet Gateway is employed during the Pre-Simulation phase. Simulation software resident in disk files on the Primary Support node is downloaded to the distributed computers during Test Preparation, under the command of the Operator Control software.

The Interconnection consists of the Proteon ProNET star-shaped ring network Wire Center and interconnection cables to all of the distributed host computers in the system. The distributed computers (MicroVAX, Intel, Heurikon) each contain plug-in, host-specific boards which connect the computers to the real time EISE network via standard Proteon ring control boards.

The Hardware Interface (HWI) node is itself one of the distributed computers on the network. It allows direct connection to each of the aircraft embedded computers via extender cards, and is used to download Operational Flight Program changes to the individual embedded computers in the system under test. In addition, it is connected to the Pilot and Control Interface, the joy-stick and rudder controls for manually "flying" the aircraft computers in a simulation.

6.3 Test Initialization and Execution

The High Speed Network is the heart of the real time simulation environment. It is a Proteon ProNET 80 megabits per second token-passing ring network allowing direct addressing of nodes to which to send data, and provides a broadcast mode enabling global simulation data to be communicated. In conjunction with custom Network Executive (NWEX), Real Time Executive (RTEX), and Simulation Data Collection (SDC) software, the EISE network provides the data path for initializing and running a simulation in the distributed environment.

Each node's physical environment is made up of a VME bus Network Interface Unit. The CPU for each NIU is a single board Motorola 68020 processor with a 16K word buffer which

allows it to transfer data at the full 80Mbit capacity of the network. The NIU processor connects to the Proteon Host Specific Board (in our application, for 68020) which in turn is connected to the Proteon Ring Control Card. The Ring Control Card contains the receive and transmit buffers for EISE network messages.

There are commercial ProNET to VME bus cards to connect to the network. Data translation to and from the node host processor's native language is performed in the NIU, and all data movement is transparent to the applications software. VME interface boards connect the node's host processor (if other than 68000 based) bus to the VME bus. These various interface boards enable the interconnection of the ProNET to Qbus for MicroVAX computers, and to Versabus, Multibus, and Harris Hbus.

A system design enhancement led to the use of standard high speed Network Interface Unit single board processors. All NIU processors interface to a 68020 Host Specific Board, moving the actual "host specific" interface from the network to the NIU. The NIU processors are then in turn interfaced to the node application processor over the Motorola VME bus.

Each NIU card cage contains a VME bus. VME-to-host interface cards serve to connect the NIU (front-end

processor) to the application host processor. Each of the application host processors is supplied with a CRT and keyboard, and each NIU also contains an Ethernet card.

The Hardware Interface application software runs on a second 68020 single board computer on the VME bus. The HWI node uses a VME-to-Multibus card to tie the Pilot and Control Interface processor to the system. The throttle/rudder joy-stick, pilot control stick, and programmable switch inputs of the PACI are processed by an Intel 80286 single board computer which also controls the simulation indicator panel lights. A VME-to-VERSAbus card connects the HWI signal box which is a VERSAbus rack of Discrete, Synchro, and AC/DC signal cards. These cards attach internally to the LRU computers through a signal breakout box, and enable bypassing analog inputs such as physical air temperature and pressure to the Central Air Data Computer and digital gyrocompass and accelerometer signals to the Inertial Navigation Unit. Individual signals can be measured from the front panel of the breakout box.

The Environment Simulation was hosted on a Harris 1200 super-minicomputer capable of executing 5 million instructions per second. A custom VME-to-HarrisBus card was developed to interface to the NIU. Because of the

unique Harris computer word size (24 bit), the superior processing power and speed of this machine was lost in the time to perform data translation for I/O to the network and the rest of the processors in the system. The software was rehosted onto a 68020 processor which attaches directly to the NIU VME bus.

A Computer Monitor and Control device can be added as a diagnostic enhancement to collect data on the internal operation of the target (aircraft system) computer when it is running its embedded program. The ability to observe Embedded Computer System operation without altering the applications software or halting the target computer makes the CMAC a powerful software maintenance and test tool. CMACs can be programmed to selectively collect target computer data. Their data collection capabilities include collecting a data value based on when it exceeds limit checks or upon an instruction/memory read, tracing register operations based on instruction access, tracing nonsequential program counter values, and issuing interrupts based on a predetermined condition.

In the programmable CMAC, data is collected through a programmable target computer interface, and stored in a buffer of approximately 4000 words. The processor unit consists of a set of processor boards programmed to scan the input buffer for data to be collected/acted upon based

on CMAC control breakpoints loaded by the operator in a breakpoint buffer. Desired output data is read by the CMAC processor section into an on-board storage buffer for later analysis.

The Operator Control NIU card cage uses a VME-to-Qbus interface to a MicroVAX-II GPX computer. The large color graphics, multi-window CRT, mouse, and keyboard are used by the operator to input, through menu selection, all simulation phase commands. The MicroVAX computer processes the application software to control the EISE.

The Displays NIU has a VME-to-Qbus interface to a MicroVAX-II processor. The MicroVAX outputs to two Megatek graphics computer systems, each programmable to produce high resolution displays. One Megatek drives a single color monitor for display of the aircraft Horizontal Situation Indicator (HSI) and Attitude Direction Indicator (ADI), while the other drives two monochrome monitors showing a Navigation "map", Parameter Plot, and aircraft Head Up Display (HUD) to allow the operator to see the current status of a simulation test. The Bus Monitor Unit (BMU) NIU has a VME-to-Qbus interface card to its MicroVAX-II host computer. The MicroVAX then connects to the 1553A Bus Monitor employing a Multiplex Bus Terminal to capture message data from the aircraft 1553 data bus.

The Simulation Data Collection host processor is also a 68020 card which plugs into the NIU VME bus. It uses its NIU Ethernet board connection to send collected data to the Primary Support host. There it is placed in mass storage for later analysis and report generation.

7. SOFTWARE REQUIREMENTS, NETWORK COMMUNICATION AND CONTROL

7.1 Non-Real Time

A group of standard software protocols, referred to as the Internet Protocol Suite (IPS), is used for non-real time communication over the EISE. They enable a user at any terminal on the EISE system to have global access to resources distributed around the network. These software protocols are "layered" to conform to the ISO Open Systems Interconnect model (Figure 7-1).

Internet Protocol (IP) is a military standard [8] which is mandatory for use in Department of Defense (DoD) packet switching networks that connect to other networks. It is at the lowest software layer of the OSI model (the Network layer), sitting logically on top of the physical network. The function of IP is to implement the distributed host addressing scheme used in the EISE.

The Internet address field consists of four, eight bit bytes. The first byte is the network field, which is important for connection via the Internet Gateway to remote LANs. (One of the design goals of the EISE is for connectivity with networks at other Air Force facilities using the Defense Data Network or DDN). The other three address bytes contain the local address of a node on the

INTERNET PROTOCOL SUITE	OSI MODEL LAYER	
FTP, TELNET	APPLICATIONS	7
TCP, UDP	TRANSPORT	4
IP	NETWORK	3
PROTEON	DATA LINK	2
	PHYSICAL	1

Internet Protocol Suite OSI Model
FIGURE 7-1

network indicated in the first address byte. There are provisions in the IP to vary the relative size of the last two address fields.

Transmission Control Protocol (TCP) is also a military standard [9] that is mandatory for use in interconnected DoD networks. It is a communication service, intended for use as a highly reliable host-to-host connection oriented transport protocol between hosts in communication networks and in interconnected systems of such networks.

On the EISE, TCP is used in conjunction with the Internet Protocol to provide internode communication in a non-real time environment. TCP is the next higher software layer (Transport) of the OSI model, just above IP. Checks are built into the protocol which uses a virtual circuit model to provide bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. TCP performs error detection, acknowledgments, sequence numbering, and flow control and maintains this information during the time a connection is open.

User Datagram Protocol (UDP) provides a simple means for non-real time message transmission between a source and destination process. It sits at the Transport layer above IP, and is a datagram oriented service. It does not guarantee delivery, lack of duplication, or proper

sequencing of messages when communicating over the network. Data integrity, however, is guaranteed.

File Transfer Protocol (FTP) sits at the Applications layer (layer 7) of the OSI model. It is used to transfer ASCII or binary files between a local and a remote host in a reliable and efficient manner, while shielding the user from variations in the file storage systems among the network hosts. This protocol relies on TCP, IP, and TELNET protocols to establish and maintain the actual network connections used in the transfer of files. Auxiliary functions are provided such as deleting or renaming files on a remote host, accessing various account directories, listing the current directory's files, and setting the data transfer modes.

TELNET's virtual terminal capability provides any user with the ability to log on to any host on the network from any terminal associated with the network. This is accomplished by creating a bi-directional character transfer circuit between the user's host and a remote host. Once the connection is complete the user's terminal appears hard-wired to the remote host, and the user has the full privileges and capabilities of a user hard-wired to the remote host. Like FTP, TELNET sits at the top Applications layer, and works in conjunction with TCP, IP, and UDP.

The Network Executive is the EISE interface used to configure and initialize the EISE from the Operator Control Station (OCS), while providing setup and status reporting of the simulation environment on all workstations and real time simulation software, prior to real time. The Network Executive performs centralized control over the non-real time network operations. It runs at the Operator Control Station and uses the TELNET protocol. The NWEX has a command interpreter which enables the operator to communicate with many different operating systems on the network using a single set of commands.

Network Executive interfaces to the Internet Protocol Suite, Test Definition, and Test Preparation. The NWEX is composed of the following major functions:

1. The Program_Interface provides a standard programmer "entry point" for calls into the NWEX functions. It allows control over service processing using a standard input/output interface between a requested service and the requester.

A command line input by the operator is parsed using "C" language argv/argc functions, with the command and arguments stored in an array. An integer status is returned to report if the command was successfully processed, the processing failed at the remote node, or the network

connection failed. It will also display a message returned from the remote process.

2. Node_Readiness determines the operational state of a node on the EISE network. Based on the command line input parameters from the operator, it will determine the type of node for establishing the communication protocol. Remote hosts use the TCP, while NIU processors communicate with NWEX using IP.

For a remote host, a connection will be established and a node name will be designated for each open connection, if it does not already exist, between a TELNET server process on the remote host and NWEX. NWEX communication is over the Ethernet network to the real time host processors in the network. If a connection already exists, an 'Are You There' message will be sent to verify that the connection is valid, and that all of the processing nodes are able to communicate across the EISE network. A status message is output to the operator at the OCS.

The NIUs are accessed via an Ethernet connection to the Internet Gateway, and then over the Proteon network. The NWEX uses the IP (UDP) to send a message to an NIU, and the NIU responds by switching the source and destination address of the message, returning it to the OCS.

Part of the Node_Readiness processing involves creating a mapping for each remote node. TELNET command data and a table specifying socket information, such as type and protocol, are stored. This information will later be used to translate the download, transfer, and terminate node NWEX commands to the remote node's operating system commands.

3. Program_Download commands transfers of executable object files from any EISE support processor to its associated target processor. The operator at the OCS specifies the support host and pathname of the file to be transferred and also the target node and destination file pathname. NWEX sends the command between the IPSs of the OCS and the target node. The file is downloaded into the real time operating system of the destination node, using the target node's operating system commands, and the target node will also be given a command to begin execution of the file immediately following. Communication between the support and target nodes is via their respective Internet Protocol Suites, using TCP for hosts and IP (FTP) for NIUs.

4. The File_Transfer function is used to command the transfer of data files between a support host and any remote target processor. The success or failure of the operation is returned in a status word to the OCS. The

interface is between IP suites of the two processors and uses the File Transfer Protocol. The operator specifies source and destination as in program download, and the transfer command is translated to operating system commands of the support node (host or NIU) then sent over the Ethernet.

5. Start_Program uses the Ethernet connection between the IPSs of the OCS and each real time node to start the execution of a program on any remote node connected to the EISE. The remote node type will be identified from the operator input, and the appropriate operating system (O.S.) commands will be transmitted over the network. A status message will be sent to the OCS to indicate success or failure.

6. The Terminate_Node function will close the TELNET connection between NWEX and a remote host, after translating the operating system commands to log off of the remote host. Status of the operation will be displayed at the operator station.

7.2 Real Time

The Real Time Protocol (RTP) is the sole communications protocol running during a simulation. It comprises layers three through five of the OSI model (Network, Transport, and Sessions), functionally sitting on top of the Physical and Data Link layers which make up the Proteon network. The Transport layer header includes a Data Control Block (DCB) which specifies data type and destination data port.

The Real Time Executive serves to control and synchronize the processing in the EISE host and all other simulation nodes. It includes a network executive for control, sequencing, and synchronization, and a node executive to control and sequence operations within each node.

The RTEX code is distributed in all of the real time nodes involved in simulation: Operator Control, Displays, Environment Simulation, LRU Simulation, Simulation Data Collection, Remote Terminal, 1553A Bus Monitor, and Hardware Interface. There are five categories of interfaces to RTEX:

1) Test Preparation sends global variable data to RTEX, and an initial value for the abort_on_overflow indicator.

2) The Operator Control station sends phase control data in the SIMCOM buffer to execute commands, and RTEX returns an acknowledgment in the OUTCOM buffer that is sent by Simulation Data Control. RTEX will also move error reports from SIMCOM and queue them to be processed at the Operator Control (OC) node.

3) RTEX sends control information to Simulation Data Control for SDC to send data or to wait for a synchronization signal. It will call the NIU_Command function of SDC to start the real time clock at the beginning of the simulation phase, call the SDC Send_and_Wait function at the end of each cycle (or phase) and the SDC Beginning_of_Cycle function at the start of each cycle. It will also invoke the SDC Wait_for_SYNC function. RTEX receives error condition indications from SDC through common memory.

4) RTEX sends control information to sequence the simulation routines, phase identification to indicate transition to a new phase, the local cycle count, and a Stop_Simulation flag to abort the simulation, and receives a Stop_Simulation flag upon an error call from a routine as well as Error Handler parameters.

5) The last interface is between SDC and the simulation routines via RTEX. RTEX updates the master

cycle count in SIMCOM to be sent to all of the simulation routines by SDC.

RTEX is composed of the following major functions:

a. Simulation_Initialization - This function consists of creating and mapping, through virtual addressing, the global common areas required on each node for all data that must be communicated between the simulation software, and building schedule tables of all routines to be sequenced for the Pre-Simulation, Simulation, and Post-Simulation phases.

The routines to be sequenced are defined in a data file which is part of the source code for RTEX. This file contains the routine name, phase identification, starting cycle, period (frequency of execution), and dependency upon other routines. When RTEX is compiled, the virtual address of each routine is linked into the executable code. First the phase identification variable will be set in OUTCOM to indicate to the simulation routines the start of initialization. Then, from the information in the data file, Simulation_Initialization will build the table of routines to be sequenced, create and map global common simulation data areas for variables that must be communicated between nodes, and call the Sequence_Routines function of RTEX to sequence the initialization routines

which will start simulation processing, initializing any local variables. Simulation_Initialization will then bring the simulation to a steady state, update the phase identification variable in OUTCOM to indicate to the OC when the initialization phase has completed, and call the Error_Handler function of RTEX to send any information on errors that may have occurred during initialization to OUTCOM, and finally call the SDC Send_and_Wait function.

b. Phase_Control - This function interfaces to the Operator Control, accepting phase transition commands in the Simulation Common (SIMCOM) to change operating phases. There are four commands:

1) (Re)Start Initialization - This command will be different for each different type of real time node machine type. Each node on the network will be successively initialized.

2) Start Pre-Simulation - Processing commands to nodes will be generated by the Sequence_Routines function of RTEX as during the real time simulation phase. Any software emulations (software models) of the target hardware will be run until they reach a steady state.

3) Start Simulation - this will invoke the Cycle_Initialization and Sequence_Routine functions of RTEX. All nodes will continue processing until their local

stop_simulation flag is set.

4) Stop Simulation, begin Post Simulation - this will set the local stop_simulation flag at each of the nodes. The Sequence_Routine function of RTEX will be called to sequence post-simulation routines, after which all network connections will be closed and RTEX will exit.

Phase_Control effects the transition to any of the four simulation phases and provides the capability to stop or abort the simulation if requested. The OC phase command is read from SIMCOM, and Phase_Control will update its phase identification variable before making the transition. For a Start Simulation phase command, Phase_Control will call the NIU_Command function of SDC to start the real time clock, and subsequently call the Cycle_Initialization function of RTEX at the beginning of each new cycle. For either the Start Initialization, Pre-Simulation, or Simulation phases commands, Phase_Control will call the Sequence_Routines function of RTEX. After the routines have been sequenced, its local phase identification variable will be written to OUTCOM to tell the OC that the phase or cycle has completed. It will then call the Error_Handling function of RTEX to dequeue any error reports and move them to OUTCOM, then call the SDC Send_and_Wait function to send the data and wait for a SYNC

message. For a null command from the OC, the SDC Wait_for_SYNC function will be called. If the command from the OC is Stop Simulation, Phase_Control will set the local stop_simulation flag. This flag may also be set by any routine detecting a fatal error condition. At this point, the post-simulation routines will be sequenced, the new phase identification will be set in OUTCOM to indicate completion, and the SDC Send_and_Wait function will be called to send the data to the OC. Phase_Control will then exit.

c. Sequence_Simulation_Routines - This function involves a table driven mechanism to sequence routines in any of the phases. The appropriate section of the sequencing table is determined by reading the local "phase ID" variable at each node. The non-real time simulation sequencing simply calls each routine in the table order. For real time simulation sequencing, routines in the local node are called based on cycle count and selected frequency, comparing the current simulation cycle with the table value. Certain routines may also have execution dependencies in the cycle, i.e., their sequencing is contingent on another routine first completing. A completion flag is set by each routine. This function executes every simulation cycle and requires a maximum of 20 microseconds of CPU time.

d. Cycle_Initialization - This function is called by Phase_Control in the Real Time Simulation Phase. It increments the cycle count and calls the Beginning_of_Cycle_Processing function of SDC to perform the required updates (initializations) at the start of each new simulation cycle. There will be a check of the NIU error flag as well as the abort_on_overflow indicator (set during Pre-Simulation) and the cycle_overflow flag, indicated in the status register of SDC. Any simulation errors occurring will be reported by calling the Error_Handler function. In addition, the stop_simulation flag will be set to tell Phase_Control and the simulation routines that a fatal error has occurred. SDC is called to move data for the next cycle from the network common to the simulation common. This executes every simulation cycle and takes no more than 0.1 msec of CPU.

e. Error_Handling - This function reports errors which occurred in the Initialization, Pre-Simulation, Simulation, or Post-Simulation phases. It accepts error parameters which include the master cycle count from SDC, and an error code and source identifier from the function experiencing the error. It will then format an error report, and queue it locally at the node. An error reporter subfunction will

move all r4ports occurring in the cycle from the local queue to the common data area OUTCOM. An error collector subfunction sends the error reports to the Operator Control node's SIMCOM data area. At the OC node, error reports will be queued locally to be processed for display to the operator by the Error_Announcer function of OC. During real time, this error reporting executes every simulation cycle and takes no more than 1.0 msec to execute.

Simulation Data Control provides a service for control of periodic transfer of selected simulation data between nodes of the EISE system. It determines, during each simulation cycle, when in the cycle real time messages are sent, as well as the content of the messages and their destinations.

SDC also performs hardware discrete processing which allows other software modules to set, clear, and read any discrete, and provides a hardware override capability for discrete indicators and switches (switch array and switch display) at the Operator Control Station.

Simulation data which needs to be sent on the network includes any data that will be referenced by more than one distributed software module. Simulation data is selected for periodic transfer prior to a real time simulation (test definition), and may be changed while the simulation is in progress. All nodes in which SDC operates are synchronized

to the master simulation cycle governed at the Operator Control Station by the master Real Time Executive.

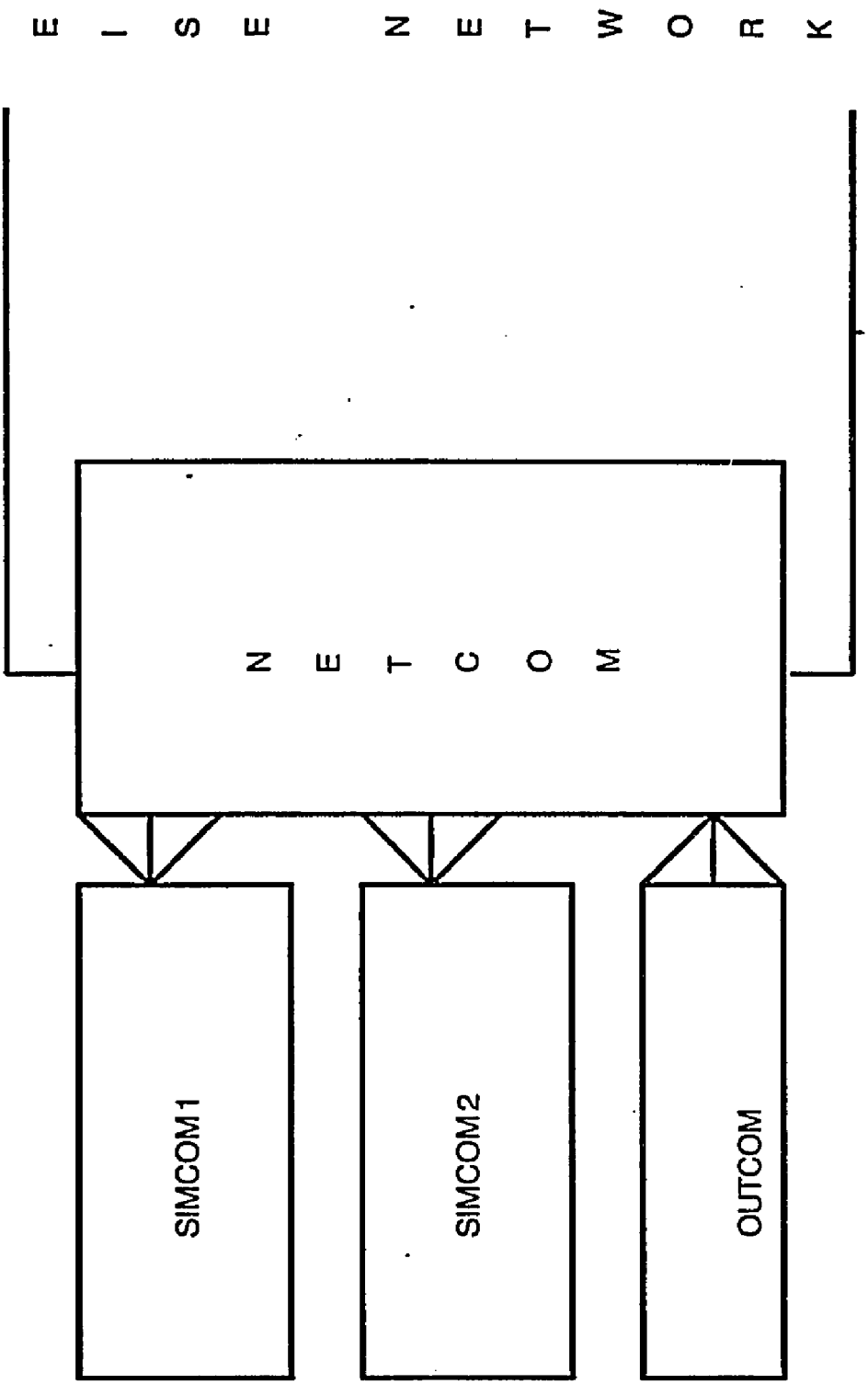
SDC provides functions to transfer selected simulation data between nodes of the EISE, and also performs discrete (switch) processing enabling software control and override of physical or graphics display switch settings from the Operator Control station. SDC resides in all nodes involved in simulation, and its functions are distributed between the Network Interface Unit and the host processor at each node. Data is selected for cyclic transfer during Test Definition, prior to running a simulation.

SDC interfaces with all of the simulation programs through SIMCOM and OUTCOM, and interfaces to the network through NETCOM (Figure 7-2). In addition, there is a common memory for hardware discrete values referred to as the Local Input Discrete Array (LIDA) and the Local Output Discrete Array (LODA).

There is one Master node running SDC, which generates SYNC messages at the beginning of every real time cycle to trigger SDC in all of the other Slave nodes.

SDC NIU Functions

The SDC functions in the NIU are Data_Receive, Receive_Interrupt, Convert_and_Send, Send_Interrupt, Command_Interrupt, Clock_Processing, Initialize_NIU and



NETCOM, SIMCOM, OUTCOM
FIGURE 7-2

Initialize_Hardware. A detailed description of the functions and their interrelationships follows.

a. The Data_Receive function moves simulation data, sent by other nodes on the EISE network, from the Proteon receive buffer (NETCOM) to the host memory (SIMCOM). For hosts other than 68000-based, Data_Receive will convert the incoming data to host format before placing it into the host memory. The function executes once for each message received from the network.

Data_Receive monitors a message source pointer table in the Proteon for incoming messages. It then uses message definition tables (set up by Initialize_NIU_Tables) to determine where to move data for this node. Each cycle it will write to an alternate SIMCOM double buffer and set a flag indicating the current SIMCOM buffer after each SYNC message or clock interrupt. Messages that are received less frequently than every cycle will be written to both buffers.

Data_Receive triggers the host operating system to wake the Wait_for_SYNC function when Clock_Processing sets a timeout flag, or when a SYNC message is received. A node configuration table (also set up by Initialize_NIU_Tables) records the host type, which tells SDC whether or not data conversion is needed, and also the frequency with which to

move data if less than every cycle.

Data_Receive reports network error information to the Real Time Executive via a status flag. After each message in the pointer table has been processed, control is transferred to the Convert_and_Send function.

b. Receive_Interrupt executes every time a message arrives from the network. It responds to hardware interrupts from the Proteon, and will build a table of pointers to messages received and held in the NETCOM buffer, to be used by Data_Receive. Receive_Interrupt will reset the local clock when a SYNC message is received, and also ensure that the Data_Receive processing is completed for that cycle. If an overflow of the NETCOM receive buffer is detected, Receive_Interrupt disables the Proteon hardware receive function until the next cycle clock timeout, and reports the error to RTEX.

c. Convert_and_Send processes commands as they are placed in the NIU Command queue by the NIU_Command or Command_Interrupt functions. A Table Initialization command will result in a call to Initialize_NIU_Tables, and a Start Clock command will cause a call to Clock_Processing. For Send Message commands, the message ID is used in conjunction with the message definition table (set up by Initialize_NIU_Tables) to ascertain the location

of the message in the host OUTCOM memory and its data structure. For a host type other that 68000, the message will first be converted to the network format, otherwise it is placed directly into the Proteon output (transmit) buffer. Convert_and_Send will command the Proteon hardware to send the message on the network. The message status buffer address is written to shared memory to be used by the Send_Interrupt function. After each message is sent, control is returned to the Data_Receive function.

d. The Send_Interrupt function monitors the status of messages sent from the node. It uses the message status buffer address from Convert_and_Send to write the status for each message sent. It will set an error flag if an error occurs in the transmission (the message was refused by the receiving node), and, if retries have been specified for this message, request that the Proteon hardware retransmit the failed message.

e. Command_Interrupt is called by the NIU_Command function. It operates in the NIU to move NIU Command IDs from processor memory into the NIU command queue.

f. Clock_Processing monitors the real time clock for a "timeout", occurring at the end of each cycle, in each of the NIUs and has two separate functions. Clock_Processing

will check the master/slave flag in the node configuration table set up by Initialize NIU Tables. In the Master SDC node, it will call the Convert_and_Send function to broadcast a SYNC message over the network. In a Slave SDC node, Clock_Processing sets a timeout flag for Data_Receive and RTEX.

Every cycle in the master SDC node, and after every timeout in the slave SDC nodes, the clock is reloaded with the clock period set up in the node configuration table. The clock at a node will be restarted after receiving a start clock command from a Convert_and_Send function call.

g. Initialize_NIU_Tables sets up the SDC message definition and control tables in the NIU during the Pre-Simulation phase, using initialization data passed from a file in the host through OUTCOM. The function is called by Convert_and_Send to read initialization data (table ID, node configuration table, message control table, and message definition table) from host memory and convert it to SDC table format. Once the tables are built, it will trigger the Wait_for_SYNC function (in the host) to synchronize with the Initialize_Host_Tables function.

h. Initialize_Hardware is performed once at hardware bootup in both NIUs and hosts. It sets up node and bus

addresses for later use in data transfer.

SDC Host Functions

The SDC functions running in the hosts are Data_Map, NIU_Command, Send_and_Wait, Wait_for_SYNC, Discrete_Input, Master_Discrete, Initialize_Host and Initialize_Hardware. A detailed description of the functions and their interrelationships follows.

a. The Data_Map function operates in the host to transfer data elements between memory locations, according to a data mapping table set up by Initialize_Host_Tables, during each simulation cycle.

b. NIU_Command operates in the host to trigger the NIU_Convert_and_Send function to begin operation. Commands IDs are passed from application software programs and the Initialize_Host_Tables and Send_and_Wait functions. The NIU_Command function places the command IDs directly into the NIU command queue in NIU memory for most machine types, or may require the Command_Interrupt function as an intermediary.

c. The Send_and_Wait function is responsible for communicating data between EISE nodes, and executes at the end of every cycle. The send (message control) table at

each node, set up by Initialize_Host_Tables, specifies the initial cycle and rate which messages are sent. Send_and_Wait first calls the Discrete_Input function, then determines which messages need to be transferred for the current cycle. It calls the NIU_Command function with a command ID for each message to be sent. After execution, Send_and_Wait will call the Wait_for_SYNC function and suspend processing until the next cycle.

d. The Wait_for_SYNC function is performed once at the end of each simulation cycle. The function will be triggered by the host (upon a call from NIU_Data_Receive or Clock_Interrupt). If the SYNC present flag is set, Wait_for_SYNC will set the cycle overflow error flag, otherwise it will wait for the next trigger from the host. This routine acts as a rendezvous mechanism.

e. Discrete_Input operates in the host to receive requests from the simulation software to set, clear, or write new discrete values. It formats discrete change requests on request and discrete change data from the Local Input Discrete Arrays each cycle. All changes are queued in the SDC change array in OUTCOM for use by the Master_Discrete_Processing function.

f. Beginning_of_Cycle_Processing operates in the host to

output discrete values to the simulation software. In response to a read call, it returns a discrete value from the Master Discrete Array to the calling simulation program. This function also transmits discrete change data to Local Output Discrete Arrays, using the Local Discrete Array Mapping Tables once each cycle. Beginning_of_Cycle_Processing will call the Hardware_Override_Display function if operating in the OC or PACI nodes.

g. The Hardware_Override Display function operates in the host to supply either the current discrete information (Master Discrete Array) or the current hardware override information (Hardware Override Array) to the Operator Control Interface and the Hardware Interface via the Local Output Discrete Array. The Master Discrete Array contains a bit indicating whether the hardware override or normal display mode is enabled.

h. Master_Discrete_Processing operates once every simulation cycle to process discrete change requests from the Discrete_Input function. All updates to discrete values are made in either the Master Discrete Array, the Hardware Override Array, or both. In the case of rotary switches, where each switch position is represented by a bit, Master Discrete Processing has error checking to

ensure that one and only one position is on at a time.

Logic is performed based on the Hardware Override (HO) bit value and the software component requesting the discrete change. Change values from routines that control the physical OCS switches and the display images of physical switches will always be used to update the Master Discrete Array. When the HO bit is TRUE, the Hardware Override Array is updated with change values received from these same software routines. Change requests originating from other software components in the system will be ignored if the HO bit is TRUE. If this bit is FALSE, then the requested discrete change will be made to the Master Discrete Array.

i. Initialize_Host_Tables is also performed at Pre-Simulation to read initialization tables (node configuration table, mapping table, message control table, message definition table, and local discrete array mapping table) from the host file system and set up control tables in the host. It will pass the NIU initialization tables through OUTCOM to Initialize NIU Tables using the NIU command function. It will then call the Wait_for_SYNC function to receive a completion indication from Initialize_NIU_Tables.

j. Initialize_Hardware is performed once at hardware bootup in both NIUs and hosts. As in the NIU, it sets up node and bus addresses for later use in data transfer.

k. The function Define_Tables is performed off-line in the Primary Support Node. It lets the user specify the contents of the SDC node configuration tables, data mapping tables, message control tables, message definition tables, and local discrete array mapping tables. All data is entered into files which will be stored in the particular node's host file system.

7.3 Data Manager

The Data Manager architecture is a standard data base with a structured query language. It functions as the collector of simulation data, and can be used for data entry and retrieval.

The Data Manager performs three main functions. Data_Loading is the user-friendly interface for data entry and retrieval, and does error checking of entered data. It is used in all phases of the simulation. Report_Generation supplies software documentation, in accordance with Military Standard 2167A. Language_Include_Files creates simulation data structure files in Ada, C, and FORTRAN languages.

8. SOFTWARE REQUIREMENTS, SIMULATION TESTING

8.1 Test Setup

The Test Setup function consists of two halves: a Test Definition tool which is a set of routines used by the test engineer to create and modify test data files defining a simulation; and a Test Preparation tool which uses these files as inputs to initialize the simulation environment by configuring the system software and hardware just prior to running a simulation.

Test Definition is an off-line process which may be executed at any time by the test engineer. It resides on the VAX 8650 Primary Support node, and is written in Ada running under the VMS operating system. The ASCII test data files which are created by Test Definition are read, prior to running the real-time simulation, by Test Preparation.

Test Definition allows the user to create a set of test data files in two ways. Existing test files may be copied and modified, or an entirely new test file may be created. A set of standard test files are available to give the operator a "baseline" test which may be tailored to a particular application. The Test Definition routines are modular and menu-selectable so that they may be separately invoked during Test Preparation if incremental

simulation changes are desired at that time. The operator has the option to print and display both Test Definition data files and Test Specification files. There is a File Consistency Check function to ensure that a complete and compatible set of all necessary files exists. Test Definition will also check for parameter interdependencies in a particular test specification file data set. The function is run automatically whenever a modification is made to a test specification file.

The types of simulation test data files created by Test Definition are the following:

A Test Specification file containing a list of Test Definition Data file names, the contents of which collectively define the EISE system initialization parameters for a particular simulation test.

The individual Test Definition files, uniquely named for each test, containing:

An Operator Control Station configuration definition file specifying which keyboards and monitors are used by Operator Control (OC), the graphic image and viewport information, and parameters defining each node in the simulation configuration

A Displays definition file which specifies the

initial display configuration, display format (tabular, graphical), and the selection of SIMCOM parameters to be displayed.

A Simulation Global Common definition file of control tables for Simulation Data Control, allowing initial values of simulation global data (SIMCOM) to be broadcast over the network.

Control tables for Simulation Data Recording, specifying which simulation data is to be recorded and the frequency/criteria. A similar function is the Bus Monitor Definition for specifying which 1553A data bus messages will be recorded.

Environmental (Aircraft) Model definition to set initial values of aircraft parameters (altitude, direction, airspeed).

Mission Planning definition for initializing discrete switch settings.

Mission Data control tables for a simulation test Flight Plan of up to 20 steerpoints specifying heading, altitude, and airspeed.

Remote Terminal Definition, to specify which LRU software models are to be used for a particular test.

The Test Preparation function runs under the Real Time

Executive and is duplicated on all processing nodes involved in the simulation. The master Test Preparation function, initiated at the Operator Control Station, performs pre-simulation initialization of the EISE in all of the network nodes involved in real time simulation.

The distributed processing system running slave Test Preparation functions include the following nodes:

1) Hardware Interface to the LRUs,
2) the Remote Terminal 1553 Bus Controller,
3) the 1553 Bus Monitor,
4) Simulation Data Collection,
5) Environment Simulation,
6) LRU Simulation,and
7) Displays

This Test Preparation function running at the OCS has a file distribution interface (FTP) to the Development and Support system to retrieve the specification and definition files. After accepting a test specification filename from the test engineer, Test Preparation will read the data file names contained in the file, after which the specified data files are moved from the Primary Support node. The individual files are downloaded to their respective slave

nodes, where distributed Test Preparation functions read the data files and initialize that node's simulation environment. Each slave node will report back to the OCS upon initialization completion. The master node will command the simulation programs to run to a steady state through as many cycles as necessary to initialize the simulation commons with a consistent set of data. When Test Preparation has terminated, a message is displayed at the OCS reporting completion of the pre-simulation initializations. The EISE is now ready to transition to the simulation phase.

8.2 Simulation Models

The software simulation models create the real-world and system environments for the A-10 avionics LRUs, the embedded computers under test. They operate on a set of data inputs and provide a set of outputs to the rest of the system, using both the MIL-STD-1553 data bus and the token passing ring network. The models function is divided between the Aircraft and Environmental model and the Inertial Navigation Unit model.

The Aircraft model performs a real-time simulation of aircraft flight dynamics including aerodynamics, the aircraft engines, primary (elevators, ailerons, rudders)

and secondary (flaps, speed brakes) flight control systems, navigation, and automatic steering (computed engine and control surface commands, approximating inputs from the aircraft pilot). The flight dynamics of the generic, six-degree-of-freedom software model [10] are tailored to the particular aircraft (the A-10 in our case) by table data describing the aircraft's physical characteristics.

The Environment simulation models physical world elements which have an impact on the aircraft flight dynamics. The model computes atmosphere (atmospheric pressure, temperature, density, local speed of sound, and wind as a function of altitude above mean sea level), gravitational acceleration based on a mean sea level radius of the earth, and magnetic variation correction.

The Inertial Navigation Unit software model is an emulation of the A-10 INU. It processes MIL-STD-1553A data bus communication and external (switch) communication to provide a realistic interface to the other components of the A-10 Inertial Navigation System (INS), as though the hardware INU were in place communicating over the system bus. Modeling the INU allows the avionics system to be "flown" on the ground. A subset of the complete INU model is a simulation model of only its Inertial Measurement Unit (IMU), which contains the gyrocompasses and accelerometers

of the Inertial Navigation Unit. The IMU model interfaces internally to the INU computer, providing digital signals indicating the orientation and speed of the aircraft.

The Master Bus Controller (MBC) is not modeled, since it is tied so closely to the operation of the avionics system.

The Central Air Data Computer is given pressure and temperature inputs from an external compressor apparatus.

The remaining two LRUs in the A-10 avionics system can be used in tandem with their software model counterparts. The test operator can choose whether to use an LRU or its software model based on the nature of the simulation. The Heads Up Display and Control Display Unit are more convenient to use running as models from the Operator Control Station, where all of the input/output takes place, though they can just as readily be used on the Hot Bench.

8.3 Data Recording

Data Recording serves to collect EISE broadcast simulation data, selected ECS input/output from the HWI node, and 1553 Data Bus messages from the BMU node. It is made up of four major functions: Initialization, List_Selection, Data_Collection, and Buffer_Management. The parameters to be collected and selection criteria are defined in lists as part of the Test Definition file

preparation, and the data recording is further controllable in real time from the operator console by selecting or deselecting different pre-defined lists.

During Test Definition, the Initialization function creates tables that are used during the simulation phase. It opens and writes a header record to each of the data recording files. List_Selection records the "logic" conditions under which the recording of lists of parameters is to be enabled. Parameter data can be recorded at regular intervals such as sub-multiples of simulation cycles, or can trigger on a certain event such as a parameter value going out of a predetermined range. Data_Collection is the real-time portion of Data Recording. This function collects and stores parameters for each enabled list, reading data values each cycle from global memory. Data_Collection operates in conjunction with a Buffer_Management function which, as the name implies, manages a pool of memory buffers used to store the recorded lists of data before they are written to disk.

8.4 Post-Test Data Analysis / Quick Look

A Data Analysis tool will provide the capability of converting raw data recorded during a simulation to a format more easily understood by an analyst. Data may be

formatted by time, as changes occur, by parameter, as a parameter crosses a threshold value, or at periodic intervals. The desired data analysis options may be specified either interactively or in batch environment by the operator. Additional sorting capability is available using a commercial data base to which selected data can be transferred for processing.

The Data Analysis function can create a report on from one to sixteen simulation data files, using a file of data analysis report specifications entered by the operator. It will extract, correlate and reformat recorded MIL-STD-1553 Bus Monitor Unit messages, simulation common (SIMCOM) data broadcast over the EISE network, and ECS input/output and trace data, from files written during a test by the Simulation Data Recording function.

A subset of the complete Data Analysis function is called Quick Look. The Quick Look tool is used immediately after a test, in an interactive mode. The capability of looking at raw test data is provided, to enable the operator to determine what to keep for further analysis. Quick Look can store (in Mass Storage or a Data Base) or delete any simulation data files, as instructed by the operator. Commands are available to move data from the Data Collection node to a central location and to form a

concatenated data file for future analysis.

Quick Look is also a report preparation function which allows the test operator to review recorded simulation data. The data for review can be sent to a CRT, file or a printer. Quick Look allows creation of the report format, including parameter selection and selection criteria. From the Main Menu selection, the operator can run the Report Format creation, Header Report which extracts information contained in header records of all the simulation data recording files, and Quick Look Report which generates the report in the requested format.

9. CONCLUSIONS

9.1 Current Constraints

The goal of the EISE project was two-fold. The first and easier of the two goals was to provide integration support capability for the A-10 aircraft. Had this been the only goal, the design would have been very straightforward. An ISF could have been built specifically for development, test and integration of software for the existing A-10 avionics. A single super-minicomputer would have been dedicated to the dynamic simulation of A-10 flight characteristics and emulation of selected hardware. This super-minicomputer would have been hard-wired to the aircraft computers, and would have had a custom operator interface that could have resembled the A-10 cockpit. The software would have been written in a mix of FORTRAN, JOVIAL, and Assembly languages to take advantage of existing simulation software already developed for other aircraft such as the F-111 and thereby minimize software development costs.

Because of an extensive upgrade planned for the A-10 avionics, and the need to be able to support two configurations of the aircraft during the phase-in period of the enhanced version, we needed to be able to support essentially two aircraft. By extension, this illustrates

the support facility problem of designing unique software and hardware for each system to be tested. In our Air Force laboratory we support three models of the F-111 on three independent test systems. While there is much overlap between the functions and features of these three F-111s, they each have individual update cycles and unique capabilities. Each of these aircraft has its own custom cockpit and separate minicomputer system to drive the simulation. The only resource sharing between the three is on the software development system, which is remotely located from the simulators.

The duplication and underutilization of expensive hardware resources pointed out the necessity for our second and more important design goal, which was to produce a reconfigurable, general purpose or "generic" software support facility. To that end, this project demonstrated that modular design and resource sharing require more planning in the design stage. For example, the software model of the A-10 aircraft, while it could have been coded specifically for the A-10, was designed to be a general-purpose four engine fixed wing aircraft despite the fact that the A-10 has only two engines. The characteristics of the A-10 are read by the model from a table that can describe any fixed wing aircraft of up to four engines.

The limitations of a single, main processor were perceived at the time of the initial design in 1983, after which a distributed host processing system was proposed. The A-10 avionics system is relatively simple compared with the avionics of more advanced tactical aircraft. The state of the art was moving at that time toward distributed processing, and with growth capability in mind the design was formulated around multiple processors. The six distributed processors plus the interconnection network met our processing requirements for roughly half the price of a single super-minicomputer system.

Our original intent was to prove that the system could operate with a network of heterogeneous processors. The final configuration is a combination of Motorola, Digital Equipment, Intel, Sun, and Symbolics computers. We learned several valuable lessons in networking non-homogeneous computers. The first is the need for fast, efficient data translation. We allocated this task to a dedicated microprocessor in each NIU which is also responsible for moving data off of and onto the network. A second lesson learned was the advantage of using global data to be shared among the processors as opposed to addressing different data for different processors. This makes the system design much simpler, as the individual processes can reside

anywhere without changing the data communication scheme. We used a replicated memory structure, where the global data is synchronously updated each cycle and each processor accesses its own copy of the data.

Over 40 percent of the present EISE system software is written in Ada, even though the project was begun before the language standard was published. In addition, the current Department of Defense standard for software development and documentation, DOD-STD-2167, was published in the middle of our project design and development phase.

With the current emphasis on stricter software engineering principles, we adopted the DOD-STD-2167 development methodology for the EISE project. We realized how hard it is to retrofit a piecemeal design into a system/segment specification structure. In traditional (pre-object oriented) software design, too little time is spent in proportion to the overall development. This leads to potentially big problems later in the integration stage. In addition, traditional (pre-Ada) software languages do not always map particularly well to the problem space.

Ada is the natural language of choice when embarking on most medium to large scale software projects. Its strongly-typed, object-oriented structure enforces modern software engineering principles on the software coder. The language is set up such that objects and types are defined

to take on a set of values, operations on those objects are defined, interfaces among objects are established, and finally the objects and operations are implemented. In addition, Computer Aided Software Engineering (CASE) tools help to bridge the gap between the pictoral representation of the program (data flow diagrams) and the software implementation. As objects and types are defined, they are compiled into a Data Dictionary. Thus a program-wide view of all of the objects used in the program is available to the programmer. Consistency checking features of the language insure proper interfaces between parts of the software.

9.2 Future Work

The desire was to be able to reuse software that was developed for this project to support test and integration of future systems. The use of the Ada language was a big factor in helping us to modularize the software components. The payback from using Ada was that the modularity is a part of the language syntax. With our VAX VMS operating system, we had a relatively simple task integrating Ada with other languages such as C, FORTRAN, and Assembly languages. As the EISE and its software components are refined and revised, more will be written in Ada.

Another feature of the Ada language is increased portability between various types of computers over previous languages. This is highly machine-dependent and assumes that the programmer has not used any special low-level features that are specific to a certain computer. An alternative to rehosting software onto a new machine is using a cross assembler on the same development computer system, which allows targeting the executable software for different machine types. In sum, more use of the Ada language should allow us to realize a cost savings in the design of follow-on ISFs.

An approach to the data communication which will be explored in the future is to limit the data transfers to only data that has changed in the cycle. In our application this wasn't necessary, but in a more complicated system, "filtering" the data could represent a significant time savings.

A specialized Artificial Intelligence (AI) or "expert system" computer will be used on the EISE as a design aid and resource allocation manager. Other projected applications are for test setup, as a diagnostic aid, and for test data reduction.

As an automated aid to design EISE networks, the expert system can translate user requirements into EISE system

specifications. The resource allocation manager can assign the shareable network resources to interface to multiple target embedded computer systems. The test setup and execution aid can present the operator with recommendations for specific test parameters and the data to be collected. As a diagnostic aid, the expert system can perform automatic failure detection by monitoring the network and its nodes, and direct testing and maintenance of failed parts of the system. In the test data reduction area, the expert system is capable of interpreting data using pattern recognition techniques, to chart a flight maneuver profile for example from instrument readings. The EISE represents a first step for the Air Force towards standardization of support facilities. The incremental cost to develop new support systems should be only a fraction of the present cost to design a unique facility. The "time sharing" of low use computer equipment will also save money. The use of the Ada language offers tremendous potential for accumulating a library of reusable software packages performing functions common to embedded computer system support.

APPENDIX I - LIST OF ACRONYMS

ADI	Attitude Direction Indicator
AI	Artificial Intelligence
ASCII	American Standard Code for Information Interchange
AISF	Avionics Integration Support Facility
BMU	Bus Monitor Unit
CMAC	Computer Monitor and Controller
CPU	Central Processing Unit
CRT	Cathode Ray Tube
CSMA/CD	Carrier Sense Multiple Access/Collision Detect
CTL	Ring ConTroL card
DCB	Data Control Block
DMA	Direct Memory Access
DoD	Department of Defense
ECS	Embedded Computer System
EISE	Extendable Integration Support Environment
EMI	Electro Magnetic Interference
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
HO	Hardware Override
HSB	Host Specific Board
HSI	Horizontal Situation Indicator
HWI	HardWare Interface
HUD	Head Up Display
I/O	Input/Output
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
INU	Inertial Navigation Unit
IP	Internet Protocol
IPS	Internet Protocol Suite
ISF	Integration Support Facility
ISO	International Standards Organization
LAN	Local Area Network
LIDA	Local Input Discrete Array
LODA	Local Output Discrete Array
LRU	Line Replaceable Unit
MBC	Master Bus Controller
MBT	Multiplex Bus Terminal
MIL-STD	Military Standard
Mbps	Megabits per second
MIPS	Million Instructions per Second
ms	milliseconds
NETCOM	Network Common (memory)
NIU	Network Interface Unit
nsec	nanosecond
NWEX	Network Executive

APPENDIX I - LIST OF ACRONYMS (continued)

OC	Operator Control
OCS	Operator Control Station
OFP	Operational Flight Program
O.S.	Operating System
OSI	Open Systems Interconnect
OUTCOM	Output Common (memory)
PACI	Pilot and Control Interface
RAM	Random Access Memory
RTEX	Real Time Executive
RTP	Real Time Protocol
SDC	Simulation Data Control
SIMCOM	Simulation Common (memory)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
usec	microsecond
VHSIC	Very High Speed Integrated Circuit

LIST OF REFERENCES

[1] Reference Manual for the Ada Programming Language
ANSI/MIL-STD 1815A-1983 February 17, 1983

[2] Military Standard - Defense System Software Development
DOD-STD-2167 4 June 1985

[3] Air Force Logistics Center Regulation 800-21

[4] MIL-STD-1553 Designer's Guide - 1982
ILC Data Device Corporation

[5] The Ethernet A Local Area Network
Data Link Layer and Physical Link Layer Specifications
Intel Corporation September 30, 1980

[6] Proteon Model p1500 VME Local Network System
Installation and Programming Guide April 1986

[7] Defense Data Network
Defense Communications Agency publication

[8] Military Standard - Internet Protocol
MIL-STD-1777 12 August 1983

[9] Military Standard - Transmission Control Protocol
MIL-STD-1778 12 August 1983

[10] Military Standard - Six Degree of Freedom Motion System Requirements for Aircrewmember Training Simulators MIL-STD-1558 22 February 1974