IN-62
Vol. 1
116471
p- 140

NASA Contractor Report 189632, Volume I

Advanced Information Processing System: The Army Fault Tolerant Architecture Conceptual Study

Volume I- Army Fault Tolerant Architecture Overview

R. E. Harper, L. S. Alger, C. A. Babikyan, B. P. Butler, S. A. Friend, R. J. Ganska, J. H. Lala, T. K. Masotto, A. J. Meyer, D. P. Morton, G. A. Nagle, C. E. Sakamaki

The Charles Stark Draper Laboratory, Inc.
Cambridge, MA

Contract NAS1-18565
July 1992

NASA
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665-5225

This page intentionally left blank.

Executive Summary

Digital computing systems needed for Army programs such as the Computer-Aided Low Altitude Helicopter Flight Program and the Armored Systems Modernization (ASM) vehicles may be characterized by high computational throughput and input/output bandwidth, hard real-time response, high reliability and availability, and maintainability, testability, and producibility requirements. In addition, such a system should be affordable to produce, procure, maintain, and upgrade.

To address these needs the Army Fault Tolerant Architecture (AFTA) is being designed and constructed under a three-year program comprising the Conceptual Study, Detailed Design and Fabrication, and Demonstration and Validation phases. This report describes the results of the Conceptual Study phase of the AFTA development. The scope of the Conceptual Study was quite broad and covered topics ranging from mission requirements to architectural synthesis and analysis to life cycle cost modeling.

AFTA is a militarized version of the Fault Tolerant Parallel Processor (FTPP) developed by the Charles Stark Draper Laboratory, Inc. AFTA is a hard-real-time Byzantine resilient parallel processor which is programmed in the Ada language. It supports testability and redundancy management strategies which permit the dynamic reconfiguration of processing sites to enhance sortie availability and mission reliability. It is composed largely of Non-Developmental Items to reduce the development risk and cost and to facilitate upgrades. Extensive analytical models and predictive verification and validation techniques are provided with AFTA to allow application designers to engineer a configuration for specific missions with a high degree of confidence that the fielded configuration will meet the mission requirements. As a part of AFTA, a fault tolerant data bus (FTDB) is being developed to provide a highly reliable, fault tolerant networking system between AFTA and other digital systems. The conceptual design of the FTDB covers many aspects of network design, including media technology, media access control, topology, routing, OSI protocol stacks, and fault detection and recovery. In addition to these traditional network topics, the FTDB also encompasses techniques from the area of fault-tolerance, including Byzantine resilience and authentication protocols.

AFTA's architectural theory of operation, the AFTA hardware architecture and components, and the architecture of the AFTA Operating System have been defined during the Conceptual Study, as well as a test and maintenance strategy for use in fielded AFTA installations. A format has been developed for representing mission requirements in a manner suitable for first-order AFTA sizing and analysis. Preliminary requirements have been obtained for two Army missions: a rotary winged aircraft mission and a ground vehicle mission. An approach to be used in reducing the probability of AFTA failure due to common-mode faults has been developed, as have analytical models for AFTA performance, reliability, availability, life cycle cost, weight, power, and volume. A plan has been developed for verifying and validating key AFTA concepts during the Dem/Val phase, especially those which cannot be cost-effectively validated by accelerated life cycle testing. The analytical models and partial Army mission requirements developed under the Conceptual Study have been used to evaluate AFTA configurations for the two selected Army missions. To assist in documentation and reprocurement of AFTA components, VHDL is used to describe and design AFTA's developmental hardware. Finally, the requirements, architecture, and operational theory of the AFTA Fault Tolerant Data Bus have been defined and described.

The next phase of the development has begun and will result in a Brassboard AFTA for demonstration and validation.

This page intentionally left blank.

Table of Contents

Executive Summary iii
Table of Contents v
List of Figures vii
List of Tables ix
Introduction to Volumes I and II xi
1. Introduction 1-1
1.1. Long-Term AFTA Development Plan 1-2
1.1.1. AFTA Reliability Growth 1-3
1.1.2. AFTA Producibility 1-5
1.1.2.1. NDI Components 1-5
1.1.2.2. Network Element 1-5
1.1.2.3. Software 1-7
1.1.3. Transition from AFTA Brassboard to Deployable AFTA 1-7
1.1.3.1. Network Element 1-7
1.1.3.2. Operating System 1-8
1.1.4. Digital Representation of AFTA Documentation 1-8
1.2. Conceptual Study 1-9
1.3. AFTA Brassboard Demonstration and Validation 1-13
1.3.1. Detailed Design 1-13
1.3.2. Fabrication, Integration, Validation 1-14
1.4. Documents Used and Generated Under This Contract 1-17
2. Requirements Definition and Acquisition 2-1
2.1. Functional Requirements 2-1
2.2. TF/TA/NOE Mission 2-2
2.2.1. Functional Description 2-2
2.2.2. Operational Scenario 2-7
2.2.3. Real-Time Constraints 2-9
2.3. Ground Vehicle Mission 2-10
2.3.1. Functional Description 2-10
2.3.2. Operational Scenario 2-10
2.3.3. Real-Time Constraints 2-12
2.3.4. SAVA Standard Compliance 2-12
2.4. Status of Requirements Acquisition 2-12
2.5. Computational Performance 2-12
2.6. Reliability and Availability 2-14
2.6.1. Sortie Availability 2-15
2.6.2. Mission Reliability 2-15
2.6.3. Vehicle Reliability 2-15
2.6.4. AFTA Reliability Formulation Approach 2-15
2.6.5. Function Reliability 2-16
2.7. Testability 2-17
2.8. Maintainability 2-17
3. AFTA Overview 3-1
3.1. Byzantine Resilience Approach to Fault Tolerance 3-3
3.2. Physical Architecture 3-8
3.3. Virtual Architecture 3-11
3.4. Communication Mechanisms 3-12
3.4.1. Voted Messages 3-13
3.4.2. Source Congruency Messages 3-15

PRECEDING PAGE BLANK NOT FILMED

PAGE iv INTENTIONALLY BLANK

3.4.3. Synchronization Message 3-18
3.4.4. Configuration Update Message 3-18
3.4.5. Synchronization 3-18
3.4.6. Byzantine Resilient Virtual Circuit Abstraction 3-22
3.4.7. Scooping 3-23
3.4.8. Input/Output Redundancy Management 3-24
3.4.8.1. Input Procedures 3-26
3.4.8.1.1. Case 1: Dumb IOC, Simplex Source in same FCR as IOC, Simplex Destination (same as source) 3-26
3.4.8.1.2. Case 2: Dumb IOC, Simplex Source in same FCR as IOC, Simplex Destination (different from source) 3-27
3.4.8.1.3. Case 3: Dumb IOC, Simplex Source in same FCR as IOC, Triplex Destination 3-29
3.4.8.1.4. Case 4: Dumb IOC, Simplex Source in same FCR as IOC, Broadcast Destination 3-31
3.4.8.1.5. Case 5: Dumb IOC, Triplex Source in same FCR as IOC, Triplex Destination 3-33
3.4.8.1.6. Case 6: Dumb IOC, Simultaneous Input Reads by Triplex, Triplex Destination 3-35
3.4.8.2. Output Procedures 3-40
3.4.8.2.1. Case 1: Simplex Source VG, Simplex IOC in same FCR as source VG 3-40
3.4.8.2.2. Case 2: Redundant Source VG, Simplex IOC in same FCR as one of VG Members, Unvoted Output Data 3-41
3.4.8.2.3. Case 3: Redundant Source VG, Simplex IOC in same FCR as one of VG Members, Voted Output Data 3-42
3.4.8.2.4. Case 4: Redundant Source VG, Redundant IOC in same FCR as one of VG Members, Unvoted Output Data 3-44
3.4.8.2.5. Case 5: Redundant Source VG, Redundant IOC in same FCR as one of VG Members, Voted Output Data 3-45
3.5. Operating System Architecture 3-46
3.5.1. Scheduling 3-47
3.5.1.1. Definitions 3-47
3.5.1.2. Requirements for Hard Real-Time Schedulers 3-49
3.5.1.3. Related Work 3-49
3.5.1.3.1. Rate Monotonic Scheduling 3-49
3.5.1.3.2. MARS (MAintainable Real-time System) 3-50
3.5.1.3.3. IAPSA 3-51
3.5.1.3.4. Reliable Computing Platform 3-51
3.5.1.3.5. NASA Space Transportation System General Purpose Computer 3-52
3.5.1.4. AFTA RG Scheduler Overview 3-53
3.5.1.4.1. Intertask Communication 3-55
3.5.1.4.2. Overview of Minor Frame 3-56
3.5.1.4.3. Preemptive Rate Group Scheduling 3-57
3.5.1.4.4. Aperiodic Hard Real-Time Task Scheduling 3-65
3.5.1.4.5. Aperiodic Non-Real-Time Task Scheduling 3-67
3.5.1.4.6. Execution of RGs on Multiple VGs 3-67
3.5.2. Fault Detection, Identification and Recovery Overview 3-69
3.5.3. Input/Output Services 3-71
3.6. The Role of Standards in the AFTA 3-72
3.7. Relationship of AFTA to the Advanced Information Processing System 3-72
Appendix A. References A-1
Appendix B. Glossary of Terms and Acronyms B-1

List of Figures

Figure 1-1. Near-Term Schedule for the AFTA Program 1-10
Figure 1-2. AFTA Conceptual Study Schedule 1-14
Figure 1-3. AFTA Dem/Val Schedule 1-18
Figure 2-1. Functional Allocation to AFTA for Helicopter TF/TA/NOE/FCS 2-3
Figure 2-2. Helicopter TF/TA/NOE Mission Scenario State Diagram 2-8
Figure 2-3. Ground Vehicle Mission Scenario 2-11
Figure 3-1. AFTA Abstract Structure 3-3
Figure 3-2. FTPP Cluster Architecture 3-9
Figure 3-3. FTPP Cluster Architecture - Sample Configuration 3-10
Figure 3-4. FTPP Sample Configuration Table 3-11
Figure 3-5. FTPP Virtual Configuration 3-12
Figure 3-6. Triplex Sender Delivers Class 1 Message x to Network Elements for Transmission 3-14
Figure 3-7. Network Elements Perform Mutual Broadcast of Class 1 Message 3-14
Figure 3-8. Network Elements Vote and Deliver Class 1 Message 3-15
Figure 3-9. Simplex Sender Delivers Class 2 Message x to Network Elements for Transmission 3-17
Figure 3-10. Network Element B Performs Initial Broadcast of Class 2 Message 3-17
Figure 3-11. Network Elements Perform Mutual Broadcast of "Reflected" Class 2 Message 3-18
Figure 3-12. Functional Synchronization 3-21
Figure 3-13. Byzantine Resilient Virtual Circuit Abstraction 3-23
Figure 3-14. Scooping a Message 3-24
Figure 3-15. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data 3-26
Figure 3-16. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data 3-27
Figure 3-17. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2 3-28
Figure 3-18. Step 3 - S1 Performs Class 2 Exchange, Delivery of Input Data to S9 3-28
Figure 3-19. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data 3-29
Figure 3-20. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2 3-30
Figure 3-21. Step 3 - S1 Performs Class 2 Exchange, Delivery of Input Data to T1 3-30
Figure 3-22. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data 3-31
Figure 3-23. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2 3-32
Figure 3-24. Step 3 - S1 Performs Class 2 Exchange to Deliver Input Data to All VGs 3-32
Figure 3-25. Steps 1 and 2 - T1 on NE0 Accesses IOC1 to Read Input Data 3-33
Figure 3-26. Step 3 - T1 Performs Class 2 Exchange, Rounds 1 and 2 3-34
Figure 3-27. Step 3 - T1 Performs Class 2 Exchange, Delivery of Input Data to T1 3-34
Figure 3-28. Steps 1 and 2 - Members of T1 on NE0, 1, and 3 Simultaneously Access IOC1, IOC2, and IOC4 to Read Input Data 3-36
Figure 3-29. Step 3 - T1 on NE0 Performs Class 2 Exchange Phases 1 and 2 3-36
Figure 3-30. Step 3 - T1 on NE0 Performs Class 2 Exchange, Delivery of Input Data to T1 3-37
Figure 3-31. Step 4 - T1 on NE1 Performs Class 2 Exchange Phases 1 and 2 3-37
Figure 3-32. Step 4 - T1 on NE1 Performs Class 2 Exchange, Delivery of Input Data to T1 3-38

Figure 3-33. Step 5 - T1 on NE3 Performs Class 2 Exchange Phases 1 and 2.......3-38
Figure 3-34. Step 5 - T1 on NE3 Performs Class 2 Exchange, Delivery of Input Data to T1........3-39
Figure 3-35. Simplex VG S9 Writes Output Data to Simplex IOC4........3-40
Figure 3-36. Triplex VG T1 Writes Unvoted Output Data to IOC4........3-41
Figure 3-37. Step 1 - Triplex VG T1 Votes Output Data, Phases 1 and 2........3-42
Figure 3-38. Step 1 - Triplex VG T1 Votes Output Data, Delivery of Voted Data to T1........3-43
Figure 3-39. Step 2 - Triplex VG T1 on NE3 Writes Voted Output Data to IOC4........3-43
Figure 3-40. Redundant Source VG Simultaneously Writes Unvoted Output Data to Multiple IOCs........3-44
Figure 3-41. Step 1 - Triplex VG T1 Votes Output Data, Phases 1 and 2........3-45
Figure 3-42. Step 1 - Triplex VG T1 Votes Output Data, Delivery of Voted Data to T1........3-46
Figure 3-43. Redundant Source VG Simultaneously Writes Voted Output Data to Multiple IOCs........3-46
Figure 3-44. Rate Group Frame - Programming Model........3-54
Figure 3-45. Architecture of RG Frames on a Single VG........3-55
Figure 3-46. Overview of Minor Frame........3-57
Figure 3-47. Rate Group Schedule - Dispatcher Task Only........3-58
Figure 3-48. Rate Group Schedule - Dispatcher+FDIR Tasks........3-59
Figure 3-49. Rate Group Schedule - Dispatcher+FDIR + R4 Tasks........3-60
Figure 3-50. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 Tasks........3-61
Figure 3-51. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 Tasks........3-62
Figure 3-52. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 +R1 Tasks........3-63
Figure 3-53. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 +R1 + Background Tasks........3-64
Figure 3-54. Scheduling of Event-Triggered Hard Real-Time Aperiodic Tasks........3-66
Figure 3-55. Phasing of RG Frames on Multiple VGs........3-69
Figure 3-56. System mode and test mode interactions........3-70
Figure 3-57. The Advanced Information Processing System Engineering Model........3-74
Figure 3-58. Interface Between AFTA and the Advanced Information Processing System Engineering Model........3-75

List of Tables

Table 2-1. Execution Times for Dynaplan Automatic Route Planner................2-5
Table 2-2. Helicopter TF/TA/NOE/FCS Requirements Summary....................2-6
Table 3-1. Maximum Event Response Latency vs. Rate Group.......................3-65

This page intentionally left blank.

Introduction to Volumes I and II

The long-term objective of the AFTA program is to develop and deploy the Army Fault Tolerant Architecture (AFTA) on a variety of Army programs such as the Computer-Aided Low Altitude Helicopter Flight Program and the Armored Systems Modernization (ASM) vehicles. Applications such as these may be characterized by a combination of computational intensiveness, real-time response requirements, high reliability and availability requirements, and maintainability, testability, and producibility requirements.

The AFTA architecture is based on the Charles Stark Draper Laboratory, Inc. Fault Tolerant Parallel Processor (FTPP). AFTA is a real-time computer possessing high reliability, maintainability, availability, testability, and computational capability. It achieves the first four properties primarily through adherence to a theoretically rigorous theory of fault tolerance known as Byzantine Resilience, through which arbitrary failure modes can be tolerated. It is designed for verifiability and quantifiability of key system attributes with a high degree of confidence, in part due to its theoretically sound basis and in part due to plausible parameterizations of fault tolerance and Operating System overheads. Through the use of parallel processing, AFTA achieves sufficient throughput for future integrated avionics and control functions. To be useful for a variety of Army applications, the number and redundancy level of processing sites in AFTA may be varied from one application to another, and AFTA is programmed in the DoD-mandated Ada language. AFTA is intended to be relatively easy to produce and upgrade through extensive use of Non Developmental Items and compliance with well-accepted electrical, mechanical, and functional standards.

Over the past few years NASA and the Strategic Defense Initiative Office (SDIO) have sponsored the Advanced Information Processing System (AIPS) program at Draper Laboratory. The overall goal of the AIPS program is to produce the knowledgebase necessary to achieve validated distributed fault tolerant computer system architectures for advanced real-time aerospace applications [Har91b]. As a part of this effort, an AIPS engineering model consisting of hardware building blocks such as Fault Tolerant Processors and Inter-Computer (IC) and Input/Output (I/O) networks and software building blocks such as Local System Services, IC and I/O Communications Services was constructed. AFTA can be considered to be a high-throughput AIPS building block which can be interfaced to the AIPS IC network. Section 3.7 describes the AIPS engineering model in more detail and illustrates how it can be interfaced with AFTA.

This report describes the results of the Conceptual Study phase of the AFTA development, and consists of fourteen sections in two volumes. Volume I is introductory in nature and contains Sections 1 through 3. Section 1 introduces the AFTA program, its objectives, and key elements of its technical approach. Section 2 defines a format for representing mission requirements in a manner suitable for first-order AFTA sizing and analysis, followed by a discussion of the current state of mission requirements acquisition for the targeted Army missions. Section 3 presents an overview of AFTA's architectural theory of operation.

Volume II contains detailed technical information and analyses in Sections 4 through 14. Section 4 describes the AFTA hardware architecture and components, and Section 5 describes the architecture of the AFTA Operating System. Section 6 describes the architecture and operational theory of the AFTA Fault Tolerant Data Bus. Section 7 presents the test and maintenance strategy developed for use in fielded AFTA installations. Section 8 describes an approach to be used in reducing the probability of AFTA failure due to common-mode faults. Section 9 develops analytical models for AFTA performance, reliability, availability, life cycle cost, weight, power, and volume. Section 10 presents the approach for using VHDL to describe and design AFTA's developmental hardware. Section 11 describes a plan for verifying and validating key AFTA concepts during the Dem/Val phase, and Section 12 utilizes the analytical models and partial mission requirements to generate AFTA configurations for the TF/TA/NOE and Ground Vehicle missions. References are contained in Section 13, and a glossary of terms and acronyms is included in Section 14.

Because some readers may wish only to read individual volumes, Volumes I and II contain some redundant information.

1. Introduction

1.1. Long-Term AFTA Development Plan

To achieve the AFTA program's long-term objective requires a multi-phased product development, production, and support cycle. A useful model for AFTA's development and deployment cycle is based on that found in MIL-STD-785B, "Reliability Program for Systems and Equipment Development and Production" [MIL-STD-785B].

First, a Conceptual Study phase is performed to ascertain the requirements of anticipated applications and develop concepts suitable for those applications. Quantitative formulations are developed for critical parameters such as performance, reliability, etc., appropriate to the level of detail available from the requirements and the proposed architectural concepts. Deliverables of this phase include a document describing the application requirements, the structure and operational theory of the proposed conceptual solution, analytical models and results used in evaluating the architecture, plans for evaluating and verifying the analytical predictions, and plans for further development phases. This documentation is provided both in hardcopy and digital format.

Next, a Demonstration and Validation (Dem/Val) phase is executed, in which the candidate solution is refined through extensive study and analysis, hardware development, test, and evaluation. In the AFTA program, a prototype of the architecture is designed and constructed from commercially available hardware, and is denoted the *AFTA Brassboard.* This prototype serves as a testbed for evaluation and improvement of the architectural concept, increases confidence in the viability of the architecture, provides information regarding the interaction of system components, and corroborates preliminary analytical and functional models. In the Dem/Val phase, the verifiable attributes of the Brassboard are investigated according to the verification plan described in Section 11 of this report, and a preliminary Failure Modes and Effects and Criticality Analysis (FMECA) is performed to identify reliability bottlenecks needing attention. The analyses produced under the Conceptual Study phase are refined based on detailed design and empirical data obtained from the Dem/Val phase, and a Full Scale Development plan is constructed. If deployable Non Developmental Items are available for use in the Brassboard, Reliability Development/Growth Testing for these items may be initiated. Deliverables of this phase include one or more copies of the Brassboard, detailed design information such as mechanical drawings, parts lists, schematics, timing analyses, data and control flow diagrams, Interface Control Documents, VHDL, ADA, and Assembler source code, hardware and software documentation, test and evaluation results, refined analytical models, the FMECA, and user/programmer guides. The documentation is provided both in hardcopy and digital format.

Upon satisfactory demonstration, validation, and refinement of the architectural concept, the Full Scale Development phase (FSD) is entered, during which the system and the principal items necessary for its support are designed, fabricated, tested, and evaluated.

The FSD phase begins with the construction of numerous plans. These include Engineering Development Model (EDM) fabrication, incoming/outgoing Quality Assurance, Environmental Stress Screening (ESS), Reliability Development/Growth Testing (RDGT), Failure Reporting And Corrective Action, Validation and Verification, Full-Scale Production (FSP), logistics, Pre-Planned Product Improvement (P^3I), and maintenance plans. A detailed Failure Modes and Effects and Criticality Analysis (FMECA) is performed to identify AFTA reliability bottlenecks. Production acceptance tests such as the Production Reliability Acceptance Test are defined. The Preliminary Design Review, Critical Design Review, and Production Readiness Review are scheduled. Deliverables from the FSD planning phase include the plans and schedule outlined above in hardcopy and digital format. Upon satisfactory completion of the FSD plans, fabrication of the EDM begins. The EDM is as far as possible identical to systems planned for Full Scale Production (FSP); for AFTA, it is constructed of military-qualified components in packages and form factors suitable for installation in the vehicles of interest. The EDM is used to verify the producibility of AFTA, undergo ESS and RDGT, and refine quantitative predictive models of AFTA attributes. Deliverables from the EDM phase include one or more EDM copies, detailed EDM engineering documentation, the FMECA, results from the ESS and RDGT, and Validation and Verification results.

After EDM testing and acquisition of detailed application requirements, the architecture is ready for Full Scale Production (FSP), in which units intended for use in one or more deployments are produced in quantity. While in use in the field, all systems (even AFTA) suffer faults and require continual maintenance, spares, and associated logistics support. During production and deployment a Failure Reporting And Corrective Action plan is exercised to identify failure modes, trace them back to weak components, and, if possible, modify the design, parts, and/or fabrication process to eliminate them. Over the AFTA's fielded life, Pre-Planned Product Improvements (P^3I) may be implemented to increase system capabilities, increase reliability/availability, and reduce support costs. It is generally expected that the field support costs will far exceed all other development and procurement costs. Finally, all systems (even AFTA) become obsolete with time, enter old age and are replaced with newer technology.

1.1.1. AFTA Reliability Growth

Under the AFTA Conceptual Study CSDL was directed to present a Reliability Growth Plan (RGP) for the AFTA development project. While it may appear premature to address such long-term issues as the Reliability Growth Plan, producibility, maintainability, and Life Cycle Costs (LCC) in the Conceptual Study phase of a development, decisions made in early phases of a program can have such a far-reaching impact on these long-term LCC drivers that they must be considered at the outset.

Product reliability growth occurs during the FSD phase as well as during the operational life cycle of the system. Under a RGP as described in MIL-HDBK-189 and MIL-STD-781D, a reliability growth curve is constructed consisting of quantitative reliability milestones and associated dates. A plan is constructed for achieving these milestones based on prior experience on similar programs and quantitative predictions.

Reliability growth for AFTA comes from a number of sources. First come increases in basic hardware component reliability. For AFTA, these components include the PEs, NEs, IOCs, PCs, backplane, enclosure, and various connectors. Component reliability increases come about through a standard program of Test, Analyze, and Fix (TAAF) and Environmental Stress Screening (ESS), in which the components are put through temperature, voltage, vibration, humidity, and other environmental conditions, suitably intensified to induce failures at a temporally accelerated rate. The induced failure modes are analyzed and their sources rectified via reengineering. This process continues until the reliability of the component meets or exceeds the milestone value, or the target value is found to be impossible to reach within a reasonable amount of time and money. This expensive and time-consuming process has presumably already been performed for the Non-Developmental Item (NDI) AFTA components. Therefore only the AFTA NEs will have to be subjected to TAAF and ESS.

Increases in AFTA reliability also come about via the insertion of reliability-enhancing technology such as VHSIC/VLSI, advanced cooling and packaging technologies, etc. into the components. These insertions can be foreseen and their effects included into the reliability growth plan as well. For example, in Section 9 of this report it is estimated that the use of VHSIC/VLSI packaging for the AFTA Network Element can increase its MTBF by over 60 percent, when compared to an implementation which uses minimal VHSIC/VLSI technology. This MTBF increase is shown in Section 12 to translate into a 30 to 70 percent reduction in the probability of catastrophic AFTA failure during the course of a one-hour rotary wing aircraft mission. Note that the reliability growth plan extends into the fielded life of the system via analysis and rectification of field failures, and via P^3Is. The AFTA RGP will indicate the anticipated reliability improvement for each major AFTA hardware component.

The next major area of reliability growth occurs in the AFTA Operating System and application software. A program similar to TAAF is used to detect and rectify software errors, although no one has yet found a way to accelerate the arrival rate of software failures by heating up and shaking the computer (although we sometimes want to!). The OS will be stressed by performing fault and error injections, varying the number of tasks over the entire declared range, performing intensive intertask communication and I/O, and subjecting the system to other stressing scenarios to be defined as the system functionality evolves. Software functions will be tested over their nominal range of inputs as well as at range boundaries, and multiple software functions will be stressed simultaneously in an integrated test program, within time and funding limitations. In addition, advanced software development and analysis technologies such as Computer Aided Software Engineering (CASE) and formal specification and verification methods play roles analogous to VHSIC/VLSI in increasing the reliability of new software and upgrades to old software. The RGP will indicate the anticipated reliability growth of the AFTA operating system and application software.

Planned improvements in component and software reliability have a direct mapping onto AFTA system reliability, availability, and LCC via the analytical models presented in this report. Through these models the RGP indicates the effect of the reliability improvements in hardware and software components on the overall AFTA reliability and availability. This can only be done through the analytical models presented in Section 9, since the reliability and availability of AFTA in its redundant configurations are expected to be sufficiently high that direct measurement of overall AFTA reliability and availability are likely to be unobtainable within a cost-effective reliability growth program. This in turn mandates that cost-effective means for verifying the meaningfulness of reliability and availability predictions be considered at the outset of the development.

1.1.2. AFTA Producibility

In formulating the conceptual design and Brassboard for AFTA, consideration must be given to the ease of producing, operating, and upgrading the fielded system. Producibility is a qualitative term which refers to the ease and cost-effectiveness with which the system can be deployed, supported, and upgraded. Simply stated, systems which are overly expensive to procure and support will probably not be deployed. It is an objective of this program to design AFTA to be producible.

1.1.2.1. NDI Components

One contribution to AFTA producibility is the use of standard interfaces to interconnect its components. An example is the use of a standard bus, such as MIL-STD-344 (SAVA), 88-VHSIC-IBM-00066 (PI-bus), and IEEE P1014/D1.2 (VMEbus), to interconnect the components (including the NE) inside an AFTA, allowing the use of Non Developmental Items (NDIs) such as processors, memories, I/O interfaces, power supplies, and other components in AFTA. NDIs are mature, tested components which are available in volume because of their use on other programs: one may presume that NDIs have fewer design errors and component frailties than new developmental items, are cheaper because of mass production, and are more producible since their producibility problems would have been ironed out before use in the AFTA program. Replacement and upgrade of these items with other NDI components with improved performance or other characteristics is simplified, as long as the new components comply with the standard bus specifications. Multi-sourced NDIs will be preferentially used in AFTA. As a safeguard against the demise of a sole-source NDI supplier, one proposed approach to ensuring NDI component availability over the long term is to come to agreement with an NDI vendor to keep all detailed design data in an escrow account; should the company ever go out of business the data will be available to the military for reprocurement from another vendor.

1.1.2.2. Network Element

The only module which must be specially developed for AFTA is the Network Element. Logical design of the Network Element is performed during the Detailed Design phase of the AFTA development. The design is also described by a VHDL behavioral model. Depending on the implementation technology of the Brassboard and the FSD AFTA, structural VHDL models may be generated as well. While expression in VHDL does not guarantee technology-independence, it is expected that such a representation will enhance the AFTA NE's producibility and reprocurement; the amount of work involved in reprocurement is a function of how much the reprocurement technology differs from the initial implementation technology (see Section 10 for details).

The Brassboard NE will contain a mixture of Surface Mount Technology (SMT) and through-hole packages. Plastic packages have the advantage of low cost and widespread availability, but are not hermetic and are therefore unsuited for hostile military environments. Reference [MIL-HDBK-217E] states that "nonhermetic parts should only be used in controlled environments (e.g., ground benign or ground fixed environments)." Therefore only ceramic parts will be used in the deployed AFTA NE. Mixed SMT/thru-hole parts have posed a slight problem for automated fabrication in the past because of the dual passes required through the soldering system, but it is felt that near-term advances in fabrication technology will overcome these problems [Biv88]. Many desirable NE parts are available in ceramic Leadless Chip Carrier (LCC) packages; however, to achieve matched thermal expansion characteristics between a ceramic LCC device and the circuit board requires the use of ceramic boards, which are expensive. Therefore LCCs and ceramic boards will be avoided in the AFTA NE. Other desirable NE parts are available in Pin Grid Array (PGA) packages. Repair of PGA devices can be difficult without the proper equipment (namely hot-air soldering stations) because of the large number of pins to be unsoldered and resoldered. It is expected that the emerging prevalence of hot air stations will decrease this concern over time, so PGA packages will be used in the AFTA NE.

The AFTA NE circuit board will be composed of polyimide with Copper traces. According to CSDL producibility engineers, polyimide provides better Copper trace adhesion than FR-4, and hence more durability and more repair cycles before pads and traces lift due to repeated solder cycles. Up to 16 signal layers are easily producible on polyimide boards; current generation NEs require only 6 layers (one power, one ground, and four signal layers), and it is unlikely that the AFTA NE will require more. Given a surfeit of layers, the NE can be designed for enhanced electrical and other characteristics. For example, signals that drive edge-sensitive inputs can be routed on a dedicated plane sandwiched between power and ground planes to prevent crosstalk and glitches. If radiation hardness ever becomes an issue, the top and bottom layers of the board can contain pads only, thus providing some shielding of the traces. Existing NEs reside on PC boards having 10 mil traces with 10 mil spacing; this geometry is easily producible with current printed circuit card technology. Since the fielded AFTA NE will be conduction cooled, a thermal management layer will be required. These layers have not been a producibility problem in the past, and can be easily formed from ground and power planes on one or more of the available signal layers.

1.1.2.3. Software

Discussions on producibility tend to focus on hardware; however, it is well known that software often dominates system life-cycle costs. Thus software producibility is at least as important as hardware producibility. The AFTA is designed to enhance software producibility through its use of the military-mandated Ada High Level Language (HLL). If Ada application code is written conservatively within the guidelines of the Ada Language Reference Manual [MIL-STD-1815], that code should be portable from one type of Processing Element to another, allowing the upgrade of AFTA PEs – possibly even changing a PE's Instruction Set Architecture (ISA) – without a major redesign of the code (some tweaking will always be necessary). In addition, the AFTA Ada application programming model is designed to be independent of the number of PEs in AFTA, the mapping of Ada tasks to PEs, or the redundancy level of a processing site. The user only specifies task-to-task communication via asynchronous message passing primitives, relying on the underlying AFTA Operating System to deliver the message to the appropriate destination task on the appropriate destination processing site(s). This transparency of task execution locale allows AFTA to be upgraded via the addition of PEs, or the redundancy level of processing sites to be changed, without affecting the fundamental data and control flow of an Ada application. In the case of repartitioning of the application code, the mapping of the repartitioned application to the modified AFTA must of course be generated and validated. To increase OS producibility, maintainability, and upgradeability, the OS is modularized such that the PE, NE, backplane bus, compiler, and kernal dependencies are well documented; where possible, the source code is modularized with respect to these dependencies as well.

1.1.3. Transition from AFTA Brassboard to Deployable AFTA

1.1.3.1. Network Element

For reasons of cost the Brassboard NE will be constructed of commercially available components. To transition the design to FSD, it must be militarized. This transition can be made relatively painlessly if it is planned for in the Conceptual Study and Brassboard Detailed Design phases. For example, the Brassboard NE will be designed to be compatible with MIL-STD-344, "Standard Army Vetronics Architecture." Conversion of the Brassboard design to a 344-compatible design will require no change in the NE's form factor, architecture, logic design, thermal characteristics, or electrical characteristics, thus facilitating this aspect of its militarization. The microelectronic devices comprising the NE must be militarized as well. This will be facilitated by the use in the Brassboard NE of circuits which are on MIL-STD-38510 slash sheets or Standard Military Drawings (SMDs). In addition, the NE's circuit timing analysis will be carried out over the entire Military temperature range of -55C to +125C to ensure that the electrical design will function over these temperature extremes. Application Specific Integrated Circuits (ASICs) used in the militarized AFTA will be implemented by constructing structural VHDL descriptions from the existing Brassboard AFTA VHDL behavioral models, and feeding the structural descriptions through logic synthesis, placement, and routing software. During ASIC fabrication the ASICs must be put through the MIL-STD-883C military qualification process before use in the deployed AFTA.

1.1.3.2. Operating System

The militarization of the AFTA Operating System is primarily an issue of how much the modules in the militarized AFTA differ from those employed in the Brassboard AFTA. For this reason, it is preferred to identify, obtain, and use the same or similar modules in both systems. This should be possible since all modules except the AFTA NE are NDI, and these items can be judiciously selected to have militarized equivalents. Failing that, it is preferential to modularize the OS to allow the easy identification and modification of software modules which depend on specific module architectures.

1.1.4. Digital Representation of AFTA Documentation

Digital computers will be heavily used in the design, fabrication, and documentation of the AFTA hardware and software. The resultant digital representation of the AFTA documentation will be provided to the Army and form the basis of a support, logistics, and maintenance digital database according to the Computer-Aided Acquisition and Logistic Support (CALS) Program Implementation Guide [MIL-HDBK-59].

The following computer platforms and software packages are being used for the construction of the AFTA Brassboard. CSDL will provide all data generated using these tools to the Army. In addition, most of the data can be converted to a digital format suitable for use by the Army's existing computer platforms and software packages, although in some cases significant loss of formatting information will occur.

General-purpose word and graphics processing are performed on the Apple Macintosh® using the Microsoft Word® and Apple MacDraw® packages, respectively. In addition to these packages, StructSoft TurboCASE® and MacFlow® are used to represent the AFTA Operating System requirements specification and detailed design. For tracking and plotting data such as parts lists, budgets, reliability, weight, power, volume, etc., the MicroSoft Excel® spreadsheet package is used. The design of the Network Element is developed using the Designworks® software package, and its smaller programmable logic devices are programmed using the Abel® package.

A Sun workstation is used to host the Vantage VHDL Spreadsheet® software package. The Sun C compiler is used for developing non-deliverable Network Element hardware test code. The Emacs editor is used as a general-purpose editor.

The XDAda® compiler for AFTA resides under the DEC VAX/VMS® environment. In addition, the DEC CMS® program is used as a basis for software configuration management. The DEC FORTRAN compiler is used for AFTA reliability, availability, and cost modeling. Either the DEC EDT® editor or the Emacs editor are available as a general-purpose editor on this platform.

The Altera MAXPLUS® software package for programming Field Programmable Gate Arrays (FPGAs) is hosted on an IBM PC or compatible.

The Conceptual Study and Dem/Val phases will now be discussed in chronological order.

1.2. Conceptual Study

The near-term objective of the AFTA program is to demonstrate and evaluate the Army Fault Tolerant Architecture (AFTA) Brassboard within the context of the Computer-Aided Low Altitude Helicopter Flight Program and the Armored Systems Modernization (ASM) Program. The subject program consists of the first two phases in the product development cycle discussed above, namely the Conceptual Study and the Brassboard Demonstration/Validation phases. These two phases are further partitioned into three separate subtasks, each spanning one year:

Conceptual Study
Detailed Design
AFTA Brassboard Fabrication and Evaluation

The approximate schedule for these phases is given in Figure 1-1:

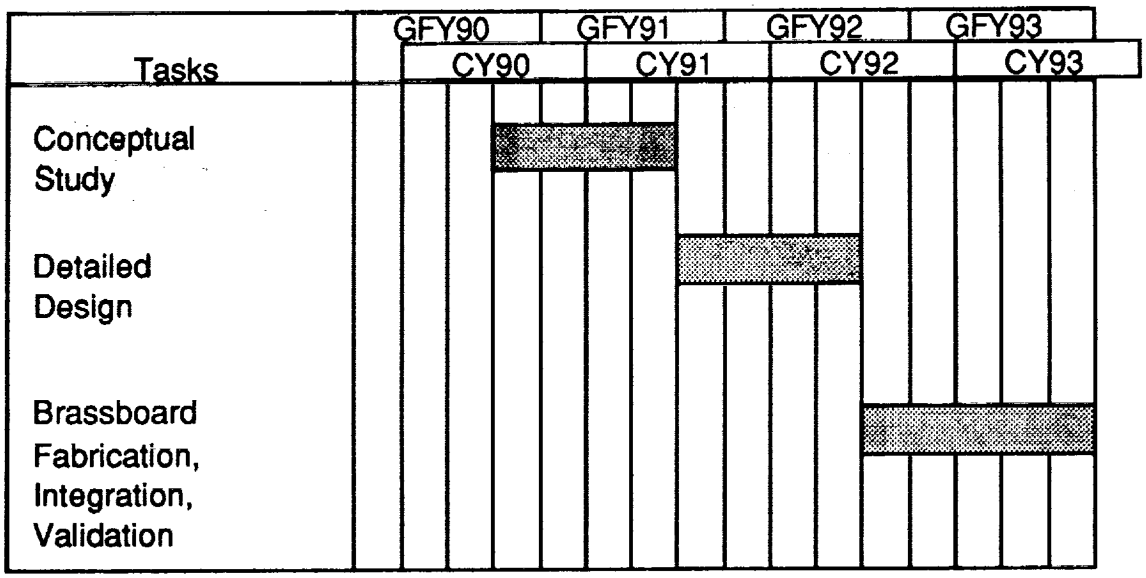


Figure 1-1. Near-Term Schedule for the AFTA Program

The current document describes the Conceptual Study phase of the AFTA program. The Conceptual Study comprises the Requirements Definition, Requirements Acquisition, Engineering Description, Analytical Modeling, Verification Plan, Architecture Configuration, the C2 Loaner, and the Fault Tolerant Data Bus subtasks.

In the Requirements Definition phase, we define a format for requirements that the application designer may place upon the computational system. Relevant requirements data include reliability, maintainability, availability, testability (RMAT), performance requirements, operational environment, mission scenario, and maintenance strategy.

In the Requirements Acquisition phase, available data are obtained for the Army missions of interest from AVRADA, CECOM-C^3, and RAMECES. These requirements are currently to be determined by the Computer-Aided Low Altitude Night Helicopter Flight and the Ground Maneuver Systems Fault Tolerant Navigation Processor programs. For brevity these applications are henceforth referred to as the "TF/TA/NOE" (for Terrain-Following/Terrain-Avoidance/Nap-of-the-Earth) and the "Ground Vehicle" applications, respectively. The requirements are transformed if possible into the format defined in the Requirements Definition phase.

In the Engineering Description phase, a detailed description is generated of the components of AFTA and how they are assembled and operated. The engineering description is sufficiently detailed to provide the analytical models with parameters such as throughput, memory, intertask communication bandwidth and latency, input/output bandwidth and latency, weight, power, size, volume, and component failure rate as a function of the architecture configuration chosen for a given Army application. In addition, the engineering description provides details on how to develop software for AFTA and operational details on fault tolerance and recovery schemes.

In the Analytical Modeling phase, analytical models are constructed to predict whether a given AFTA configuration will meet the requirements as specified in the Requirements Acquisition phase. These models are parameterized so as to be useful in estimating the characteristics of the Brassboard as well as multiple deployable AFTA configurations.

The Verification Plan phase comprises the construction of a plan for demonstrating that the analytical models predict system characteristics with reasonable accuracy. This plan is executed in the Dem/Val phase.

In the Architecture Configuration phase, the AFTA architectural parameters are adjusted to realize conceptual architectures for the two Army missions: the helicopter TF/TA/NOE mission and the Ground Vehicle mission. The analytical models developed in the Analytical Modeling phase are used to predict AFTA reliability, availability, weight, power, volume, and Life Cycle Costs for these missions.

In the C2 loaner subtask, the FTPP Cluster 2 (C2) is to be delivered to AVRADA for evaluation and familiarization with FTPP technology. The C2 is a quadruply redundant uniprocessor version of the FTPP and hosts the same basic Ada Run Time System and software development environment as AFTA. The AFTA software development environment is purchased and delivered to AVRADA in the Conceptual Study phase to jump start the AFTA application software development process. In a related effort, the testability of the C2 Network Element (NE) will be evaluated by writing and demonstrating self-test software; the lessons learned from this exercise will be used to improve the testability of the AFTA Network Element.

Common-mode faults are those which occur in more than one copy of a redundant computation due to a common source. Thus, they can defeat redundancy-based fault tolerance techniques such as those used in AFTA. A methodology for detecting and recovering from common-mode faults in AFTA will be developed. In addition, a plan for verifying the effectiveness of the common-mode fault tolerance techniques comprising the methodology will be formulated. In the event that application-specific information for the study is needed, the helicopter TF/TA/NOE application will be used as a context for the common-mode fault tolerance study.

As a separate but related effort, a fault tolerant data bus (FTDB) is developed to provide a fault tolerant networking system for AFTA and other digital systems, including the Silicon Graphics display processor, the Merit Technologies MT-1 VME system, the real-time AI system, sensor and image processors, and flight and engine controls. The objective of the fault-tolerant data bus effort is to provide highly reliable end-to-end communications between the above systems. The conceptual design of the FTDB covers many aspects of network design, including media technology, media access control, topology, routing, OSI protocol stacks, and fault detection and recovery. In addition to these traditional network topics, the FTDB also encompasses techniques from the area of fault-tolerance, including Byzantine resilience and authentication protocols.

The schedulable work modules for the Conceptual Study phase of the AFTA development are given below. The Subtasks in the parentheses refer to the Statement of Work [NAS1-18565-14] items which the work modules address.

1. Requirements acquisition and interpretation. (Subtask 1)
2. Availability and reliability models. (Subtasks 2.D.1, 2.D.3)
3. Cost model. (Subtask 6.D)
4. Weight, power, size/volume models (Subtask 2.E)
5. Performance models. (Subtask 3)
6. VHDL study. (Subtask 2.C)
7. Fault tolerant data bus study. (Subtask 2.B)
8. Fault tolerance and recovery options study. (Subtasks 2.A, 2.D.2, 2.D.3, 2.E)
9. AFTA verification plan. (Subtask 6.A, 6.B, 6.C)
10. AFTA architecture synthesis and analysis. (Subtasks 3,4)
11. FTPP Cluster 2 (C2) Operating System. (Subtask 5)
12. C2 delivery to AVRADA's hot bench. (Subtask 5)
13. AFTA software development system procurement. (Subtask 7)
14. FTPP C2 NE comprehensive self-test demonstration. (Subtask 9)
15. AFTA common-mode fault tolerance study. (Subtask 8)
16. Final Report.

The schedule for the Conceptual Study phase of the AFTA development is given in Figure 1-2.

1.3. AFTA Brassboard Demonstration and Validation

Following the completion and evaluation of the Conceptual Study phase the Brassboard Dem/Val phase begins. The first year of Dem/Val comprises the Detailed Design phase, while the second year comprises the Fabrication, Integration, and Validation phase.

1.3.1. Detailed Design

The intent of the detailed design phase is to design the hardware and software architectures recommended from the Conceptual Study phase, in preparation for Brassboard fabrication in the Fabrication, Integration, and Validation phase. It comprises the following subtasks:

The Brassboard AFTA Network Element is completely designed; the Brassboard NE contains no ASICs. In addition, a comprehensive set of NE self-tests is designed. A software simulation of the NE is constructed for use in Operating System and other AFTA software development efforts.

The design of the backplane-independent components of the NE is described in VHDL at the behavioral level.

The basic AFTA Operating System (OS) is designed and documented; the OS includes task scheduling, intertask communication, input/output services, and Fault Detection, Identification, and Recovery functions.

Common-mode fault avoidance, removal, and tolerance techniques are selected and designed from among those identified in the Conceptual Study phase.

The quantitative models of AFTA's reliability, availability, weight, power, volume, failure rate, life-cycle cost, and other parameters are refined as design and application mission details become available.

The Fault Tolerant Data Bus (FTDB) is designed.

Deliverables of this phase include detailed design information such as mechanical drawings, parts lists, schematics, timing analyses, data and control flow diagrams, Interface Control Documents, VHDL, ADA, and Assembler source code, hardware and software documentation, refined analytical models, the software simulation of the NE, and user/programmer guides. The documentation is provided both in hardcopy and digital format.

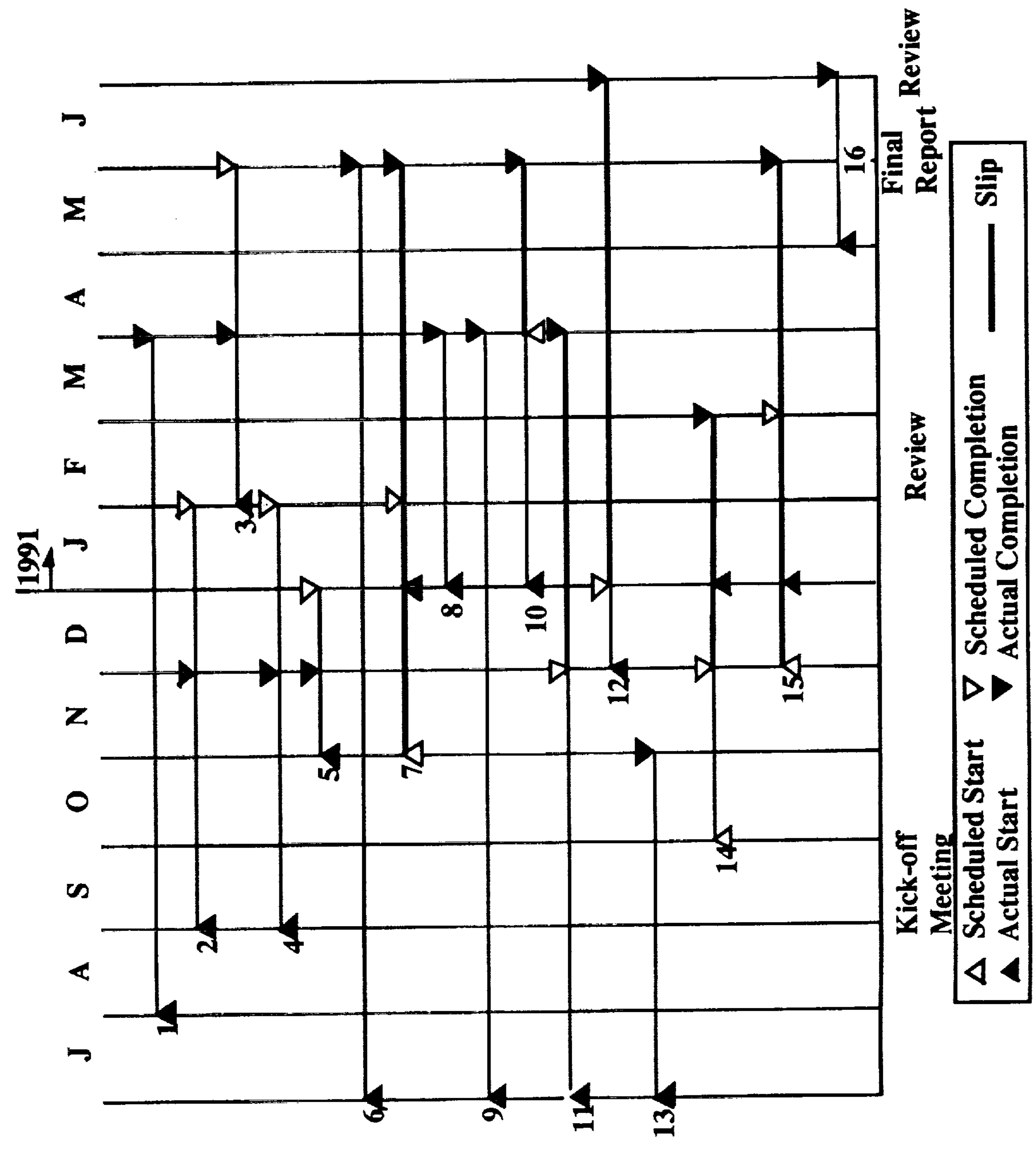


Figure 1-2. AFTA Conceptual Study Schedule

1.3.2. Fabrication, Integration, Validation

In the second year of the Dem/Val phase one or more Brassboard AFTAs are assembled. The AFTA's Network Elements (NEs) are fabricated and tested, the Processing Elements (PEs), Input/Output Controllers (IOCs), backplanes, and Power Conditioners (PCs) are purchased, and the Operating System (OS) software is completed. Fabrication of the FTDB also begins.

After fabrication and integration of the components the Brassboard is delivered to the Army for demonstration and validation. For the demonstration, a representative application is ported to AFTA. Subsequently, the critical parameters of AFTA are evaluated according to the verification plan described in Section 11. The demonstration and validation tasks are expected to be performed jointly by the Army, NASA LaRC, and CSDL.

The following parameters are measured, both with and without injected faults in relation to the TF/TA NOE application:

Delivered throughput per processing site
Available memory per processing site
Effective intertask communication bandwidth
Effective I/O bandwidth
Iteration rate of a task
Reliability†
Availability†
Testability
Cost per unit of service†
Weight, power, and volume

The fault recovery and common mode fault tolerance capabilities specified by AVRADA will also be demonstrated.

Deliverables of this phase include one or more copies of the Brassboard, detailed test and evaluation results, and refined analytical models. The documentation is provided both in hardcopy and digital format.

The schedulable milestones for the Dem/Val are as follows:

Hardware:

† Cannot be measured directly. See Section 11.

1. Complete the detailed design of the AFTA Brassboard Network Element. This includes schematics, netlists, PAL equations, microcode, timing diagrams, parts lists, and board layouts.

2. Complete the Network Element simulation.

3. Complete the fabrication and testing of a single AFTA Brassboard Network Element.

4. Complete the fabrication and testing of a redundant set of AFTA Brassboard Network Elements.

5. Complete VHDL behavioral model of the Network Element Scoreboard.

6. Complete VHDL behavioral model of Network Element Global Controller, Voter/Fault Tolerant Clock, and Ring Buffer Manager.

Basic Operating System:

1. Complete the Software Development Plan.

2. Complete the Software Requirements Specification.

3. Develop a debugging/development support environment.

4. Perform Detailed Design.

5. Code and test subset of functions with simulation.

6. Test subset of functions with AFTA hardware.

Quantitative Models:

1. The quantitative models of AFTA are updated based on refined engineering data and mission details.

The schedule for the Dem/Val hardware and software development is shown in Figure 1-3.

Additional work modules associated with Dem/Val have not been scheduled as of completion of the Conceptual Study. These tasks include the implementation of the Common-Mode Fault Avoidance, Removal, and Tolerance Techniques and the development of the Fault Tolerant Data Bus.

1.4. Documents Used and Generated Under This Contract

The documents used under the AFTA Conceptual Study Phase are listed in Section 13, "References." The documents generated under this contract are listed below.

NASA Contractor Report 189632, Volumes I and II, "AFTA Conceptual Study Final Report"

"An Ultrareliable Integrated Digital Computer for Helicopters," R. Harper, G. Grant, 10th Digital Avionics Systems Conference, October 1991

"A Fault-Tolerant Network Architecture for Integrated Avionics," B. Butler, S. Adams, 10th Digital Avionics Systems Conference, October 1991

"Hardware Modeling and Top-Down Design using VHDL," D. Morton, MIT SM Thesis, June 1991

"The Design and Construction of a Data Path Chip Set for a Fault Tolerant Parallel Processor," C. Sakamaki, MIT SM Thesis, February 1991

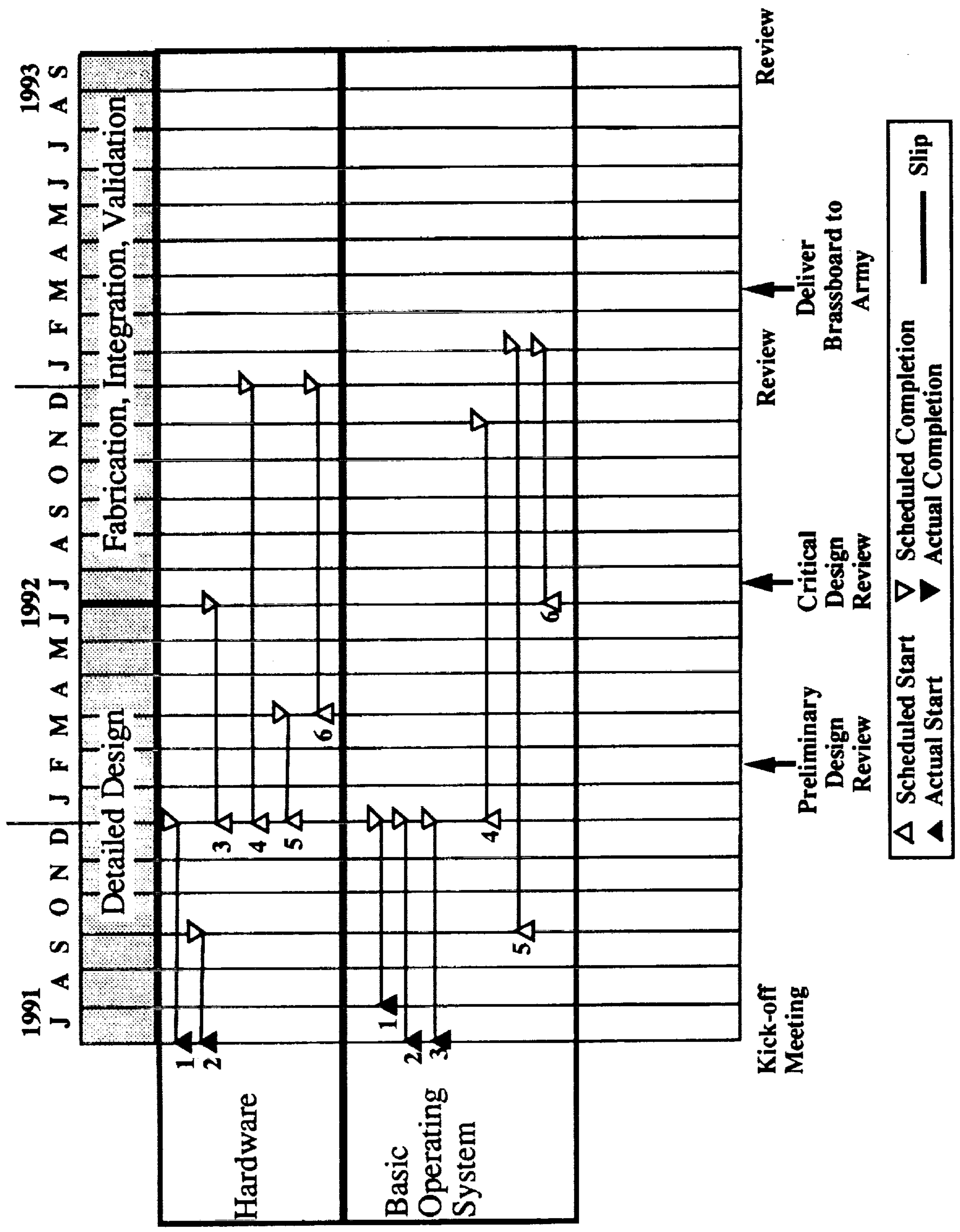


Figure 1-3. AFTA Dem/Val Schedule

2. Requirements Definition and Acquisition

This section outlines the mission requirements which were obtained during the Conceptual Study phase for the helicopter TF/TA/NOE and Ground Vehicle missions. Following the presentation of the mission requirements, a format is suggested for computational performance, reliability, and availability requirements to be imposed on AFTA by a mission designer. The eventual availability of refined and accurate requirements in this format is necessary to allow accurate architecture synthesis†, analysis, verification, and validation. In the preliminary stages of mission design, detailed parameters are unavailable, and only coarser, more heuristic measures must be used to size AFTA. Any architecture synthesis process must function with such preliminary requirements, while tracking their progressive refinement and solidification into a format such as that outlined below.

2.1. Functional Requirements

The following representative functions might be performed by the AFTA in the helicopter TF/TA/NOE and the Ground Vehicle missions. These functions are described in more detail in Sections 2.2 and 2.3.

TF/TA/NOE (TF/TA/NOE mission only)
Threat Avoidance/Engagement
Mission Planning
Guidance
Navigation
Vehicle Control
Sensor Management and Processing
Communications Management
Display Management

† In this document the term "architecture synthesis" refers to the configuration of AFTA for a specific mission via the selection of the appropriate number of Processing Elements, Network Elements, redundancy levels of processing sites, and other AFTA configuration parameters.

2.2. TF/TA/NOE Mission

The following description of the functions required to perform helicopter Terrain Following/Terrain Avoidance (TF/TA) and Nap-of-the-Earth (NOE) flight is synthesized from References [Deu88], [Pek88], [Ber90], [Boo88], and [Fel90]. The requirements obtained from these sources are insufficient to allow them to be cast into the formalism defined in subsequent sections. However, any architecture synthesis process must be able to work with incomplete, preliminary, and evolving requirements data. It is planned in the subsequent phases of the AFTA development to continue the search for detailed application data, so the subsequent section should be viewed as a first look at the application requirements.

2.2.1. Functional Description

Helicopter TF/TA and NOE night flight requires the successful execution of the flight control system, very near-field planning, near-field planning, trajectory generation, far-field navigation, sensor management, navigation, and display functions (Figure 2-1). Additional functions may include vehicle health management and propulsion control. Because of the disastrous ramifications of losing TF/TA/NOE capability during low-altitude night flight, it may be safely assumed that most of these functions are flight-critical.

The vehicle flight control system (FCS) translates acceleration and attitude commands from the pilot or autopilot into commands to the vehicle actuators. The response time scale for this function is on the order of tens of milliseconds. Reference [Osd88] describes a representative fly-by-wire FCS for the Advanced AH-64 helicopter, in which a single triplex flight control computer performs this function. Neither the type of computer nor its throughput are cited in [Osd88]. As another point of reference, the NASA Ames UH-60 FCS currently under design uses three Motorola 68030 processors† . Two are used for computation purposes and one is used for I/O. The iteration rate of the control system is 50 Hz. 200 Hz Higher Harmonic Control (HHC) is not implemented. Approximately 100 inputs and a small number of actuators are accessed by the FCS, not all at 50 Hz. The current design incorporates no redundancy since a hydraulic backup is available in the event of FCS failure. In both of these designs, part of the processors' throughput is devoted to the operating system, and in the first design to the redundancy management as well.

† Personal communication between Jay Lala and Michelle Eshow, US Army Aviation Systems Command.

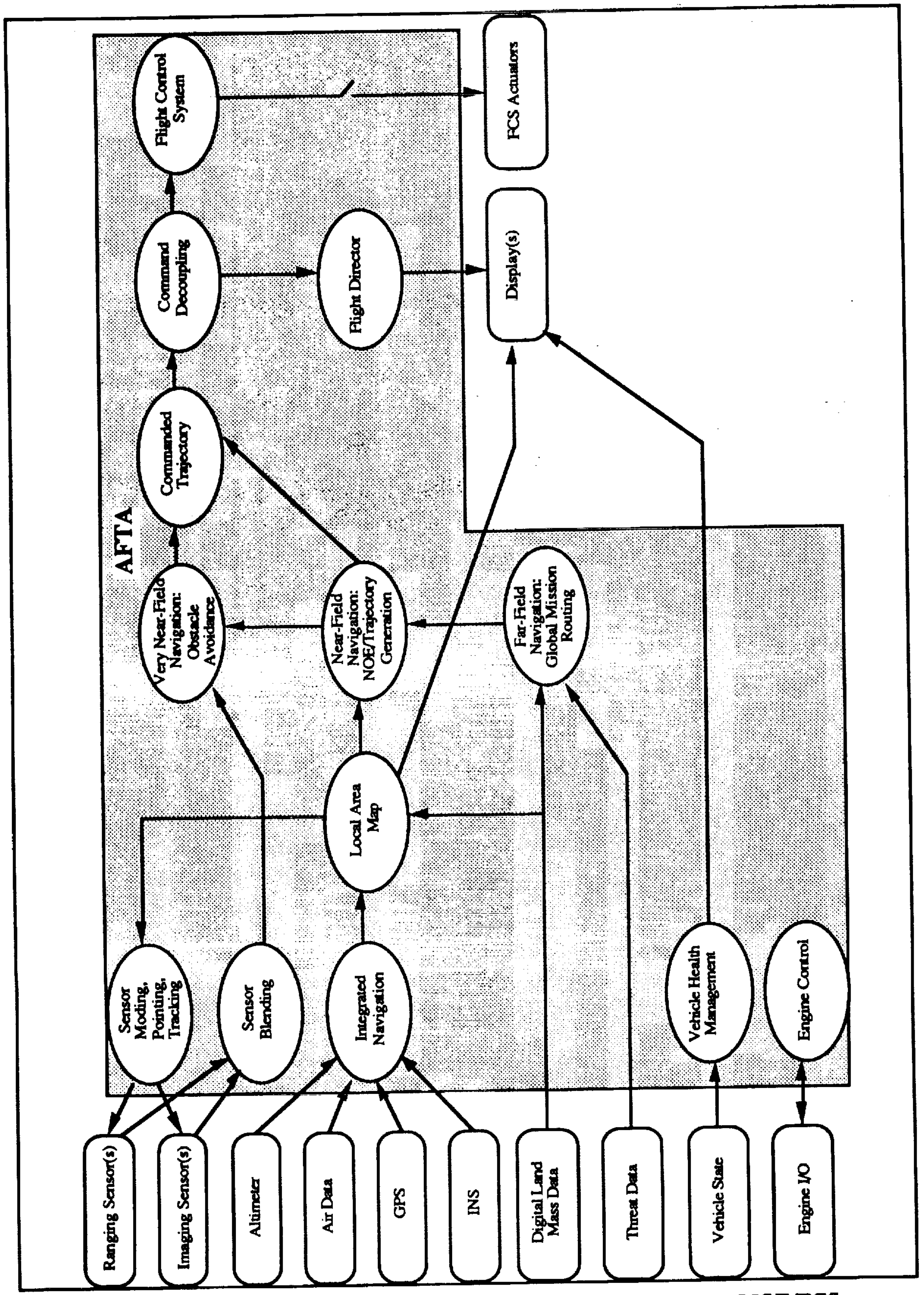


Figure 2-1. Functional Allocation to AFTA for Helicopter TF/TA/NOE/FCS

In the very near-field planning regime, the immediate environment is sensed by imaging sensors such as the crew's eyes, FLIR, radar, etc. When obstacles are detected by this sensor suite, the very near-field system may generate appropriate commands to override the trajectory generated by the near-field trajectory generation function. The time scale for this function is not documented in the available literature but is expected to be on the order of a few seconds. It also expected to make extensive use of image and sensor processing.

The near-field planning function utilizes the preplanned mission route and the local area map to generate a "locally optimized flyable trajectory over the next 30 seconds or so of flight time [Pek88]." Reference [Pek88] further states that the Dynapath near-field planning algorithm requires about 6 seconds of 1 MIPS MicroVAX II CPU time to plan 30 seconds of flight.

The trajectory generation function integrates the very-near and near-field outputs to construct a flyable six degree of freedom trajectory to accomplish specific tactical objectives, avoid obstacles, and follow the immediate terrain profile.

The far-field navigation function is responsible for the definition, prioritization, and ordering of mission goals, allocation of vehicle stores, resources, and weapons, timeline generation, waypoint generation and planning, and optimization and determination of the overall route to be flown. This function can be carried out both prior to the mission and, given sufficient in-flight computational resources, during the mission as the situation changes due to new threat information, inflight failures, unanticipated environmental conditions, a changing tactical environment, etc. The time scale of this function is the entire mission duration. In [Deu88] it is asserted that far-field planning (comprising the goal planning, waypoint path planning, and timeline management algorithms developed in [Deu88]) can be successfully performed for a flight system on a computer having a 1 MIPS throughput. As a second data point, reference [Pek88] provides execution times for the Dynaplan Mission Planning Workstation's automatic route planner (Table 2-1).

The TF/TA/NOE-specific sensor management function is responsible for providing the pilot with a reliable and accurate view of the ground via sensor selection, mode control, pointing, redundancy management, and sensor/image data processing. The TF/TA/NOE-specific sensor suite consists of the FLIR, radar [Pek88], and projected map [Fel90] subsystems.

# Map Grid Units per Side	1 MIPS Micro- VAX II Time	PC/AT Time
50	10 sec	30 sec
75	20 sec	1 min
100	40 sec	2 min
125	1.5 min	5 min
150	3 min	12 min
200	7 min	21 min

Table 2-1. Execution Times for Dynaplan Automatic Route Planner

As another point of reference for TF/TA and display requirements, the AVRADA UH-60 STAR TF/TA application[†] requires the throughput of one Motorola 68020 processor, the Pilot Vehicle Interface (PVI) requires three Motorola 68020 processors, and the Cathode Ray Tube/Helmet Mounted Display (CRT/HMD) imagery requires two MIPS, Inc. R3000 processors. In all cases part of the processors' throughput is devoted to executive and other overhead functions.

The I/O requirements are as follows:

128 Bytes at 8 Hz 1553 to/from INS/GPS
298 Bytes at 4 Hz 1553 to/from PVIs
128 Bytes at 1 Hz 1553 to/from INS/GPS
TBD Ethernet to/from R3000s

In this application, all TF/TA/NOE functionality is assumed to be flight critical.

The traditional navigation function controls and integrates the radar/baro altimeters, INS, GPS, air data, doppler radar, compass, and other vehicle state sensors to generate a reliable and accurate estimate of the vehicle state [Fel90].

The display function is responsible for providing the crew with a representation of vehicle state and systems status, steering, speed and attitude cues, a representation of the external environment, and other functions TBD. In AFTA it is anticipated that the display function will be used to provide the flight and ground crews with indications of the overall AFTA operability level and fault detection and identification results.

[†] Personal communication between Rick Harper and Bob Patel, US Army AVRADA.

In reference [Fel90], the Pave Low III Enhanced Navigation System Mission Computer provides control of the navigation sensors, radar, FLIR, and projected map for a MH-53J helicopter using one IBM AP-102 computer.

The Advanced Apache Mission Equipment Package described in [Boo88] does not perform many of the functions listed above (e.g., TF/TA/NOE/FCS), but does perform other functions which might be required of the computation system on an advanced attack helicopter. They are listed here for reference purposes. The throughput requirements are such that ten 1.0 to 1.5 MIP 1750A processors are needed: System processor*, Display processor*, Weapons processor*, Nav processor*, Power management processor*, Comm processor, Aircraft Survivability processor, Digital map processor, Air data processor, and AAWWS processor. Dual-dual 1553 buses are used for interprocessor communication, and two 2 HSDBs† are used for various subsystem communication purposes.

Throughput
1 TF/TA/NOE 680x0
3 PVI 680x0s
2 imagery R3000s
2 FCS 680x0s
1 FCS I/O 680x0
Total of 9 processing sites
I/O
200 Bytes in, ≈20 Bytes out at 50 Hz
128 Bytes in/out at 8 Hz
298 Bytes in/out at 4 Hz
128 Bytes in/out at 1 Hz
Aggregate bandwidth of 13,344 bytes/sec (0.107Mbit/sec) required
Criticality
Flight Critical

Table 2-2. Helicopter TF/TA/NOE/FCS Requirements Summary

* Duplex 1.0 to 1.5 MIP 1750A processors. The types of other processors are uncited in [Boo88].
† [Boo88] does not state whether this is the JIAWG HSDB, the SAVA HSDB, or some other HSDB.

Table 2-2 summarizes the TF/TA/NOE/FCS requirements obtained from the literature search. It is expected that, for reasons mentioned above, the processor count indicated in the table provides a throughput that significantly exceeds that needed by the application. In many cases only a fraction of the processors' throughput is actually spent on the application program; the rest is spent on executive and other overhead functions, especially for applications currently running under disk-based multitasking multiuser operating systems.

2.2.2. Operational Scenario

To serve as a context for evaluating the AFTA in the helicopter TF/TA/NOE application, an extremely simplified mission state transition model has been constructed (Figure 2-2). The model consists of four states. In the "hiatus" state, the vehicle is idle, neither undergoing maintenance nor performing a mission. Presumably the vehicle is powered down. Periodically, the vehicle is called upon to sortie. It may be assumed that it is desirable to transition from the hiatus state to the sortie state as quickly as possible, as in a scramble scenario; therefore there is only sufficient time for rapid I-BIT (See Section 5 for definitions of the various forms of AFTA Built In Test and the AFTA testability strategy) to determine AFTA's readiness. The I-BIT determines whether sufficient AFTA components are nonfaulty to permit formation of the Minimum Dispatch Complement (MDC), and thus whether the vehicle may sortie*. Note that it is assumed that the vehicle is allowed to sortie with faults. If MDC is met, the vehicle enters the "sortie" state, in which it performs its mission. If MDC cannot be formed, the vehicle is unable to sortie because AFTA is broken, and enters the "maintenance" state. The analytical models found in Section 9 of this report quantify the probability of occurrence of this event.

During the mission, C-BIT is continually executed to detect and recover from faults according to fault recovery strategies selected by the application designer from among those listed in Section 5. However, critical failure modes exist which can defeat AFTA's redundancy and its C-BIT. One example is two or more faults occurring in different members of a redundant processing site within such a small time window that C-BIT has not recovered from the first before the second arrives. Another example is a long string of faults causing exhaustion of spares, such that recovery from subsequent faults is impossible. In these cases, since it is assumed that the AFTA is performing flight critical functions, it follows that the vehicle cannot sustain controlled flight, and is lost. It enters the "vehicle lost" state.

* The vehicle may be unable to sortie for other reasons as well. For the current analysis we only consider the impact of AFTA faults on vehicle availability and reliability.

The analytical models found in Section 9 of this report quantify the probability of occurrence of this event. When the sortie has been completed without a critical AFTA failure, the vehicle is assumed to enter the post-flight maintenance state, in which the vehicle is returned to a fault-free hiatus state via the maintenance procedure outlined below.

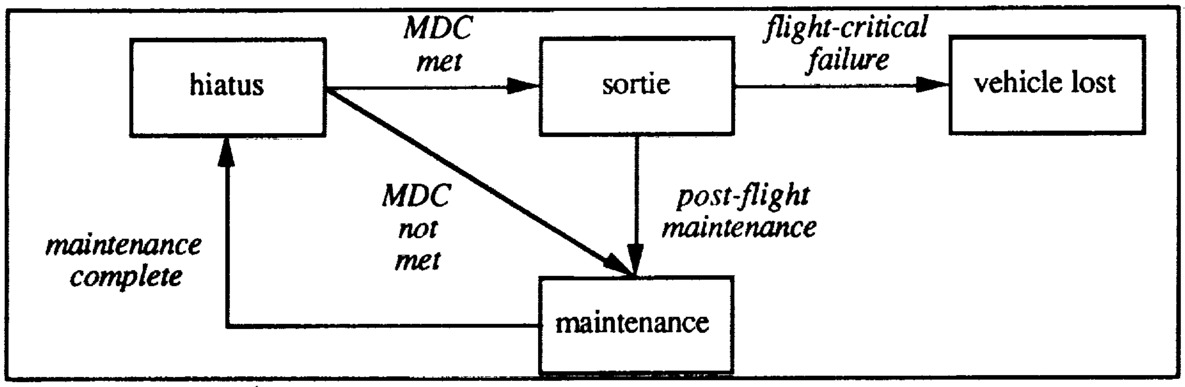


Figure 2-2. Helicopter TF/TA/NOE Mission Scenario State Diagram

In the maintenance state, the field maintenance crew executes AFTA M-BIT to identify faulty Line Replaceable Modules (LRMs) or Line Replaceable Units (LRUs). The crew also interrogates fault logs which I-BIT, C-BIT, and M-BIT have placed in AFTA's non-volatile mass memory. The crew replaces components identified as faulty, re-executes M-BIT to confirm fault exorcism, and returns the vehicle to its hiatus state (details on the AFTA maintenance plan are presented in Section 7). It is assumed that the maintenance task takes at least as long as the sortie itself, so the vehicle has essentially missed the sortie opportunity.

The expected mission environment for the helicopter TF/TA/NOE corresponds to the Aircraft, Rotary wing environment described in MIL-HDBK-217E, with expected sortie times, denoted T_s, ranging from one to four hours. The hiatus time (denoted T_h) plus the sortie time T_s is assumed to be 24 hours; this assumption is trivially changed.

$$T_h + T_s = 24 \text{ hours} \tag{2.9}$$

The hiatus environment is assumed to correspond to the Ground, Fixed environment described in MIL-HDBK-217E. Failure rates predicted assuming a Ground, Fixed environment may be higher than those experienced in the field because MIL-HDBK-217E failure rates are for powered-up components. In actuality, AFTA will be powered down until just before sortie, so the actual hiatus failure rate may be lower than that predicted by MIL-HDBK-217E. On the other hand, it is well known that cycling the power to electronics is more stressful than leaving them on. Unfortunately, MIL-HDBK-217E contains no methodology for calculating the failure rate of electronics as a function of power cycles, so the Ground, Fixed assumption will have to suffice.

The mission and hiatus times, environments, and failure rates can be trivially changed in the analytical models presented in Section 9, and can be modified upon direction from the Army.

2.2.3. Real-Time Constraints

In the helicopter application it is expected that AFTA will be executing flight-critical functions having hard real-time constraints on the order of tens of milliseconds. Function dropouts exceeding these times may be assumed to result in Loss of Control (LOC) of the vehicle. In-flight redundancy management and fault recovery options are therefore constrained to those which do not interrupt service for intervals exceeding the real-time constraint interval. One such option which will be analyzed in Section 9 as the "reliability model" is known as a "downmode" or "graceful degradation" policy, in which faulted members of a redundant processing site are quickly masked out from further computations, but no lengthy recovery attempt is made. This policy meets fast real-time constraints, but, because it does not attempt to differentiate transient from permanent faults nor switch in spare processors to restore the redundant site's redundancy level, eventually could result in exhaustion of the site's redundancy and hence loss of that site. It is therefore suitable only for mission times which are short compared to the AFTA components' MTBFs. In contrast, during pre-sortie AFTA initialization, in which no flight-critical real-time constraints exist, a significant amount of time (on the order of seconds) may be available for the AFTA redundancy management functions to attempt to optimally construct the MDC from the available nonfaulty resources. Such a policy does a much better job at determining which resources should be members of which redundant processing sites, and performing the appropriate and lengthy initializations and state transfers. This option is also analyzed in Section 9, where it is referred to as the "availability model." Since a given AFTA implementation supports the use of multiple redundancy management policies, each appropriate for a given mission phase, both of the above policies would be used in the helicopter TF/TA/NOE application.

The various fault recovery options possible in AFTA, which include the two mentioned above, are described in Section 5.

2.3. Ground Vehicle Mission

Requirements acquisition for the Ground Vehicle mission is in a more preliminary stage than for the TF/TA/NOE mission. All that has been obtained in the Conceptual Study is a general outline of the mission, a system specification for the Combat Vehicle Command and Control System [CVC2], a set of standards to which the AFTA must comply for the Ground Vehicle application [MIL-STD-344], and informal guidance from CECOM-C3. More detailed requirements will be acquired during subsequent phases of the AFTA program to permit plausible synthesis, analysis, verification, and validation of an AFTA configuration for the Ground Vehicle application.

2.3.1. Functional Description

The initial ground vehicle application for AFTA is the Ground Maneuver Systems Fault Tolerant Navigation Processor (FTNP) for Armored Systems Modernization (ASM) vehicles. These vehicles must traverse rough terrain at high rates of speed, often during combat operations. To assist the crew in this function, the Navigation Processor stores three-dimensional terrain data and digital map information; it also receives real-time position updates from a GPS and/or inertial navigation support system, real-time reports of enemy threats and targets that the vehicle will encounter on its path of travel, and local vehicle sensor system threat detection inputs. Based on threat inputs, the vehicle must either change its path of travel to avoid the threat, or engage the threat on the move. The Navigation Processor must in both cases compute and provide to the crew real-time path information that allows them to effectively maneuver across the terrain at high speed, while meeting appropriate threat avoidance, engagement, and vehicle mobility constraints. In addition, the Navigation Processor must provide the ASM gun system with real-time information needed to target threats while on the move.

2.3.2. Operational Scenario

The operational scenario for a ground vehicle differs from that of the helicopter in several respects. In addition, simplifying assumptions are made to facilitate the reuse of the combinatorial analytical models presented in Section 9. During the hiatus phase, it is assumed that the vehicle is undergoing continual maintenance via incessant tinkering by a bored crew. This motivates replacing the hiatus state in the helicopter scenario with the "maintenance" state in the Ground Vehicle scenario. Because of the continual maintenance, it is assumed for analytical convenience that the vehicle sorties into the "mission" state with zero faults and unity availability.

During the mission, cumulative failures can eventually defeat the ASM FTNP's redundancy. The results of a debilitating failure of ASM FTNP, while perhaps having severe consequences to the success of the mission, do not necessarily result in the loss of the vehicle and crew†. Therefore upon a mission-critical failure the vehicle enters the "mission failed" state. Eventually the vehicle and maintenance crew find each other, at which time the vehicle enters the maintenance state. In the event of successful mission completion, the vehicle also enters the maintenance state, where it is tested and maintained as outlined above and in Section 7. The Ground Vehicle mission scenario is diagrammed in Figure 2-3.

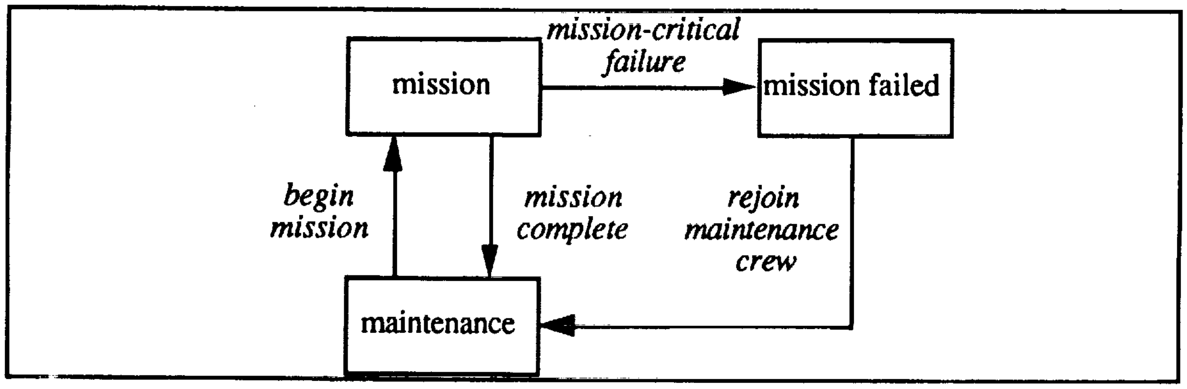


Figure 2-3. Ground Vehicle Mission Scenario

The mission times range from several hours to an indefinite interval (possibly days or weeks) whose duration depends on tactical mission requirements and ASM vehicle parameters such as reliability and availability. Based on informal information from CECOM-C3 a mission duration of 24 hours is typical. The maintenance time also varies widely. Its exact value does not enter into the FTNP reliability calculations.

The mission environment for the ASM FTNP corresponds to the Ground, Mobile environment described in MIL-HDBK-217E. As in the helicopter scenario, the maintenance environment is (irrelevantly) assumed to correspond to the Ground, Fixed environment described in MIL-HDBK-217E.

† AFTA is capable of performing safety- and vehicle-critical functions in a ground as well as in a flight vehicle. However, the worst that can happen if AFTA is executing the ground vehicle navigation function outlined above is that the vehicle gets lost and the crew has to navigate by more conventional means. This can in turn result in vehicle loss if it unwittingly strays into a threat. Changing the model's mission loss state to a vehicle loss state is a purely syntactic one for the purposes of this analysis.

2.3.3. Real-Time Constraints

As in the helicopter mission, the ASM FTNP must perform real-time functions. Unlike the helicopter mission, short ASM FTNP dropouts (e.g., one second) for fault recovery do not necessarily result in LOC. In addition, the long mission duration requires the use of a longevity-enhancing redundancy management strategy which maximizes the use of resources and does not gratuitously condemn any components. This type of redundancy management would perform permanent and transient fault discrimination, followed by recovery from transient faults or switching in of spares, and other longevity-enhancing redundancy management techniques. Such a redundancy management strategy is compatible with the relatively relaxed real-time response requirements of the FTNP, and allows the "availability" model described in Section 9 to be used to calculate the probability of mission loss.

2.3.4. SAVA Standard Compliance

For maximum commonality with existing and planned ASM computing platforms and systems, the Navigation Processor must operate on the ASM Standard Army Vetronics Architecture (SAVA) Bus [MIL-STD-344]. The Navigation Processor may be a "black box" connected to the ASM SAVA bus, or may be composed of a network of processing boards performing the navigation functions through the SAVA bus. The role of standards in the AFTA is discussed in more detail in Section 3.

2.4. Status of Requirements Acquisition

Under the Conceptual Study the requirements acquisition process was initiated for the two Army applications of interest. This process resulted in a preliminary sense of the computational functions to be performed and the operational scenarios for the two missions, but additional requirements acquisition and analysis must be aggressively pursued during the remainder of the AFTA development to ensure that the system produced under Dem/Val has relevance to the needs of future Army missions.

2.5. Computational Performance

Each function may consist of one or more computational tasks which may be scheduled on the AFTA. The computational requirements of an application are embodied in the characteristics of the schedulable tasks which make up its functions. The list presented below enumerates computational characteristics of the application's tasks which are needed to determine important AFTA architectural parameters such as the number of processors, the amount of memory, the bandwidths of the various networks, etc. In an ideal world, the entries in the list below would be provided for each schedulable task of each function. However, in the early phases of system synthesis, only aggregate figures are usually available.

Throughput. A task's throughput requirement is expressed in terms of the number of instructions per second required for a processor to execute the task within a desired period of time. A variety of instruction-mix benchmarks may be used for specifying throughput. These include the Digital Avionics Instruction Set (DAIS), Whetstones, Dhrystones, DP Linpacks, Specmarks, VAX Units Per Second (VUPS), and simply Instructions Per Second (IPS). If they are believed to be accurate representations of an application task's computational requirements, most of these quantifications are at worst dangerously misleading and at best useful only for preliminary gross system sizing. It is much more accurate to continually benchmark an application task's execution time using a specific high level language, compiler, and processing element as its development proceeds. Throughput margins are usually specified on a task-by-task basis.

Iteration rate. For many real-time control applications a task must be executed at a fixed, known frequency. This is referred to as the task's iteration rate and is expressed in Hertz. AFTA is currently being designed primarily for the execution of iterative tasks having hard real-time deadlines.

Execution latency. Often a given task must be executed and provide outputs either to another task or to an interface to the outside world within a specified time of the occurrence of an event, such as the reading of a sensor input. The time interval from the occurrence of such an event to the provision of the output is denoted the task execution latency. For an iterative task, the latency can be at most the reciprocal of its iteration rate.

Scheduling constraints. Tasks can be preemptible by other higher priority tasks or non-preemptible by those tasks. In general, a hierarchy of preemption relationships exists in which some tasks may preempt others but not vice-versa. It is preferable for verification purposes if this preemption hierarchy is static.

Task precedence and data dependency relationships. In general, tasks must execute in a precedence relationship which is defined by the needs of the application. This relationship must be specified unless the order in which the tasks execute is unimportant.

Memory. The memory requirements of each task must be specified. Usually, memory consumption margins are specified on a task-by-task basis.

Intertask communication bandwidth and latency. The intertask communications bandwidth is the number of bytes of data per second that a source task transmits to a recipient task. The intertask latency is the time in seconds between when a source task transmits a data byte and the recipient task receives that byte.

Input/Output bandwidth and latency. The input bandwidth is the number of bytes transferred per second from an input device to a recipient task. The output bandwidth is the number of bytes transferred per second from a source task to an output device. The input latency is the time in seconds between the sampling of an input byte by the input device and the availability of that byte at the input of the destination task. The output latency is the time in seconds between when a computational task generates an output byte for delivery to an output device and when the output device receives the output byte.

Transport Lag. The transport lag of a task is defined to be the sum of its input, execution, and output latencies.

In AFTA most timing relationships such as latencies are specified and measured with respect to periodic timer-generated frame boundaries, as described in Section 5.

2.6. Reliability and Availability

AFTA will reside in a vehicle which must periodically perform sorties, during which it may be called upon to perform mission- or vehicle-critical functions. The unreliability of AFTA will contribute to the probability that the vehicle cannot sortie, the probability that the mission cannot be accomplished, and the probability that the vehicle is destroyed via loss of a flight-critical computational function. The maximum allowable probabilities of occurrence of these events must be specified by the vehicle or application designer. In addition, the minimum AFTA configuration (i.e., number of processing sites, their redundancy levels, I/O device availability, etc.) required for dispatch, mission success, and vehicle survival may be specified, as well as the number of sequential or simultaneous faults the AFTA must be able to tolerate. These parameters may in some cases be inferred from the probabilistic requirements.

Three reliability-related figures of merit can be used to specify and evaluate AFTA implementations.

2.6.1. Sortie Availability

The *Sortie Availability* is defined to be equal to one minus the probability that the vehicle is prevented by AFTA faults from beginning a mission at the desired time. Usually, maintenance activities are employed during operational hiatus to maximize this quantity; these activities must be specified. Alternatively, the minimum complement of resources required to sortie, known as the Minimum Dispatch Complement (MDC) may be specified or may be determined by reliability analysis.

2.6.2. Mission Reliability

The *Mission Reliability* is defined to be equal to one minus the probability that failure of AFTA causes mission abort. It is generally assumed that during the mission no maintenance is possible. Alternatively, the minimum complement of resources required to execute the mission, known as the Minimum Mission Complement (MMC) may be specified. In this case the probability that an MMC is operational may be computed to yield mission reliability.

2.6.3. Vehicle Reliability

The *Vehicle Reliability* is defined to be equal to one minus the probability that failure of AFTA causes destruction of the vehicle. Alternatively, the minimum complement of resources required to safely control the vehicle, known as the Minimum Safety Complement (MSC) may be specified. The probability that an MSC is operational may be computed to yield vehicle reliability.

2.6.4. AFTA Reliability Formulation Approach

To successfully perform the mission, the computer system must be capable of executing numerous computational functions. In the AFTA reliability formulation approach used in this document, the application designer specifies a quantity known as function reliability; the required reliability of a given function, such as guidance, navigation, TF/TA/NOE, displays, or flight control, is specified. At the application designer's level, let the computer system S be represented by the set of functions F_j required for mission success or vehicle safety. In the sequel the computations will be set up assuming that the vehicle reliability is being calculated. The formulation for mission reliability is identical except the functions needed to perform the mission are used to compose S instead of those needed for vehicle safety.

$$S = \{F_j \mid F_j \text{ needed to maintain vehicle safety}\} \quad (2.1)$$

The probability that the computer system can maintain vehicle safety is the joint probability that all needed functions can be performed. Let this quantity be denoted by R_{sys}.

$$R_{sys} = \text{Prob}(F_j \text{ can be performed, } \forall\, F_j \in S) \quad (2.2)$$

For example, a given system may require that the Navigation, TF/TA/NOE, and Flight Control functions, perhaps denoted F_N, F_T, and F_{FC}, be available for the successful pilotage and control of the vehicle. Then

$$S = \{F_N, F_T, F_{FC}\} \quad (2.3)$$

and

$$R_{sys} = \text{Prob}(F_N, F_T, \text{ and } F_{FC} \text{ can be performed}) \quad (2.4)$$

2.6.5. Function Reliability

The system reliability hinges on the reliability with which the requisite computational functions can be performed. Function reliability refers to the probability that a given function can be performed by the computational system at a given time. In general, the successful performance of a function requires the correct operation of a suite of resources. In AFTA, a resource is considered to be a redundant processing group (known as a VG) or a set of Input Output Controllers (known as IOCs). The probability that the function can be performed is equal to the probability that the related suite of resources is operational. The application designer must enumerate the resources which must be operational in order to perform each given function. A given function may require multiple resources and a given resource may support multiple functions, subject to the computational requirements of the application. Arbitrary mappings of functions to resources and vice-versa are allowed.

For reliability evaluation purposes, a function F_j is equivalent to the set of resources required to execute it:

$$F_j = \{\text{resource}_i \mid \text{resource}_i \text{ needed to execute function } F_j\} \quad (2.5)$$

The reliability of function F_j is the joint probability that all needed resources are operational. Let this quantity be denoted by R_{Fj}.

$$R_{Fj} = \text{Prob}(\text{resource}_i \text{ operational, } \forall\, \text{resource}_i \in F_j) \quad (2.6)$$

For example, a given function, say F_N, may require the services of processing sites 1 and 2 (known as VGs in an AFTA) and I/O Controllers 4 and 5.

$$F_N = \{VG1, VG2, IO4, IO5\} \tag{2.7}$$

The probability of successful execution of this function, i.e., its function reliability, is the probability that all of these resources are operational.

$$R_{FN} \equiv \text{Prob}(F_N \text{ available}) = \text{Prob}(VG1 \ \& \ VG2 \ \& \ IO4 \ \& \ IO5 \text{ operational}) \tag{2.8}$$

When the suite of requisite computational functions and their mappings to AFTA resources are known, the analytical models presented in Section 9 can be used to estimate the mission and vehicle reliability as defined above.

2.7. Testability

A testable system provides an unambiguous indication of the existence and location of a fault. AFTA is partitioned into Line Replaceable Modules (LRMs) which are the units of field diagnosis and repair. Typically an LRM comprises one circuit card assembly containing, for example, one or more Processing Elements. Section 4 describes the AFTA LRMs and LRUs in more detail. AFTA is intended to be a testable system at the LRM level, in that a fault in AFTA will be detected and isolated to the faulty LRM. It is intended that fault diagnosis and identification be performed in the vehicle without any external test equipment, using self-test functions which are an intrinsic part of the AFTA Operating System. Depot testing will extend the testing granularity to the integrated-circuit level. The AFTA approach to testability is described in detail in Sections 5 and 7.

2.8. Maintainability

AFTA is being designed for two-level maintenance. The first maintenance level occurs at the field organizational unit, and consists of a maintenance crew which executes the various AFTA BIT suites (described in Section 5), removes and replaces LRMs/LRUs identified as faulty, replaces them with nonfaulty LRMs/LRUs, sends the faulty components back to the depot, and re-exercises BIT to confirm fault exorcism. One option being considered for AFTA is to discard faulted components instead of returning them to the depot; a decision regarding this option is pending. The second level of maintenance occurs at the depot, where received LRMs/LRUs are tested, repaired, and returned to the spares logistics pool. The AFTA maintenance procedure is described in detail in Section 7.

This page intentionally left blank.

3. AFTA Overview

The information contained in this report is intended to comprise an exhaustive description of AFTA, albeit at a high functional level of abstraction. Additional features may be added to AFTA as subsequent phases of the development program are performed.

AFTA is based on the Fault Tolerant Parallel Processor (FTPP) architecture developed by the Charles Stark Draper Laboratory (CSDL). The FTPP architecture was conceived to satisfy the dual requirements for a computer system of ultra-high reliability and high throughput. To satisfy ultra-high reliability requirements, the FTPP is designed to be resilient to Byzantine faults. This terminology will be defined below.

To satisfy high throughput requirements, the FTPP utilizes multiple Processing Elements to obtain parallel processing capability. For an operational definition of parallel processing, we invoke [Hwa84]:

> "Parallel processing is an efficient form of information processing which emphasizes exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity, and pipelining. Parallel events may occur in multiple resources during the same time interval; simultaneous events may occur at the same time instant; and pipelined events may occur in overlapped time spans. These concurrent events are attainable in a computer system at various processing levels. Parallel processing demands concurrent execution of many programs in the computer. It is in contrast to sequential processing. It is a cost-effective means to improve system performance through concurrent activities in the computer." ([Hwa84] p. 6)

The FTPP may also be considered a distributed architecture in the sense of [Cha84], which defines loosely-coupled distributed systems as

> "multi-computer configurations that do not share immediate memory and can be dispersed over wide geographical areas." ([Cha84] p. vi)
>
> "Logically, a loosely coupled distributed system can be considered to be a collection of processes running on various processor elements (or nodes). Although processes running on the same node can communicate using shared memory, processing running on separate nodes must communicate using messages (or their equivalent)." ([Cha84] p. 194)

The FTPP architecture is described in references [Abl88], [Bab90a], [Har87], [Har88a], [Har88b], and [Har91a]. It is composed of Processing Elements (PEs) and specially designed hardware components referred to as Network Elements (NEs). The multiple Processing Elements provide a parallel processing environment as well as compo-

[redacted]

nents for hardware redundancy. The group of Network Elements acts as the intercomputer communications network and the redundancy management hardware. As with most complex computing systems, AFTA is best viewed as a layered system (Figure 3-1). The top layer consists of the applications programs themselves. In an ideal world, applications are constructed by the applications engineers without regard for the parallel and redundant nature of the AFTA system. In this view, AFTA supports a virtual architecture of a number of Ada tasks which may execute in parallel, subject to preemption, data, and control flow dependencies. The tasks communicate using message passing. In reality, the applications engineers must to a large extent assist in the selection of appropriate task-to-processing site mappings, processing site redundancy levels, fault recovery strategies, and other parameters from among those made available by the AFTA architecture.

The next lower layer consists of the AFTA System Services, described in Section 5. Certain services are visible and may be invoked by the applications programmer; these include input/output, task scheduling, and intertask communication services. This layer is intended to mask the complexity of the AFTA's lower layers from the programmer. The application programmer's interface to AFTA is described in Section 5. Other important functions of the AFTA System Services are not directly accessible by the applications programmer and are performed in a manner which is largely transparent. These include the functions of mapping of tasks to processing sites, arranging of preemption of lower priority tasks by higher priority ones, routing intertask messages to remote tasks, disassembling and reassembling long messages, performing input/output functions, Built-In Testing (BIT) and fault logging, and fielding software exceptions. Other functions are fault detection, identification, and recovery (FDIR); reconfiguration of the parallel resources into redundant computing sites; and interfacing to the interprocessor communication network hardware. The application tasks and AFTA System Services execute on the AFTA Processing Elements, as indicated in Figure 3-1.

The next lower layer of the AFTA consists of the interprocessor communication network hardware, known as Network Elements. This hardware implements the interprocessor message passing functions of the AFTA. In addition, it implements throughput-critical fault tolerance-specific functions such as voting of messages emanating from redundant processing sites and providing error indications, assisting in synchronizing redundant processing sites, and assisting in arranging the non-redundant processing resources of the AFTA into redundant processing sites based on the needs of the application, mission mode, and fault state of the AFTA. The inter-Network Element communication links provide high-bandwidth, electrically isolated, optical communication paths between the Network Elements of the AFTA. The data transmissions over the links also keep the Network Elements synchronized to within a small skew using a digital phase-locked loop. A detailed description of the Network Element is provided in Section 4 of this report.

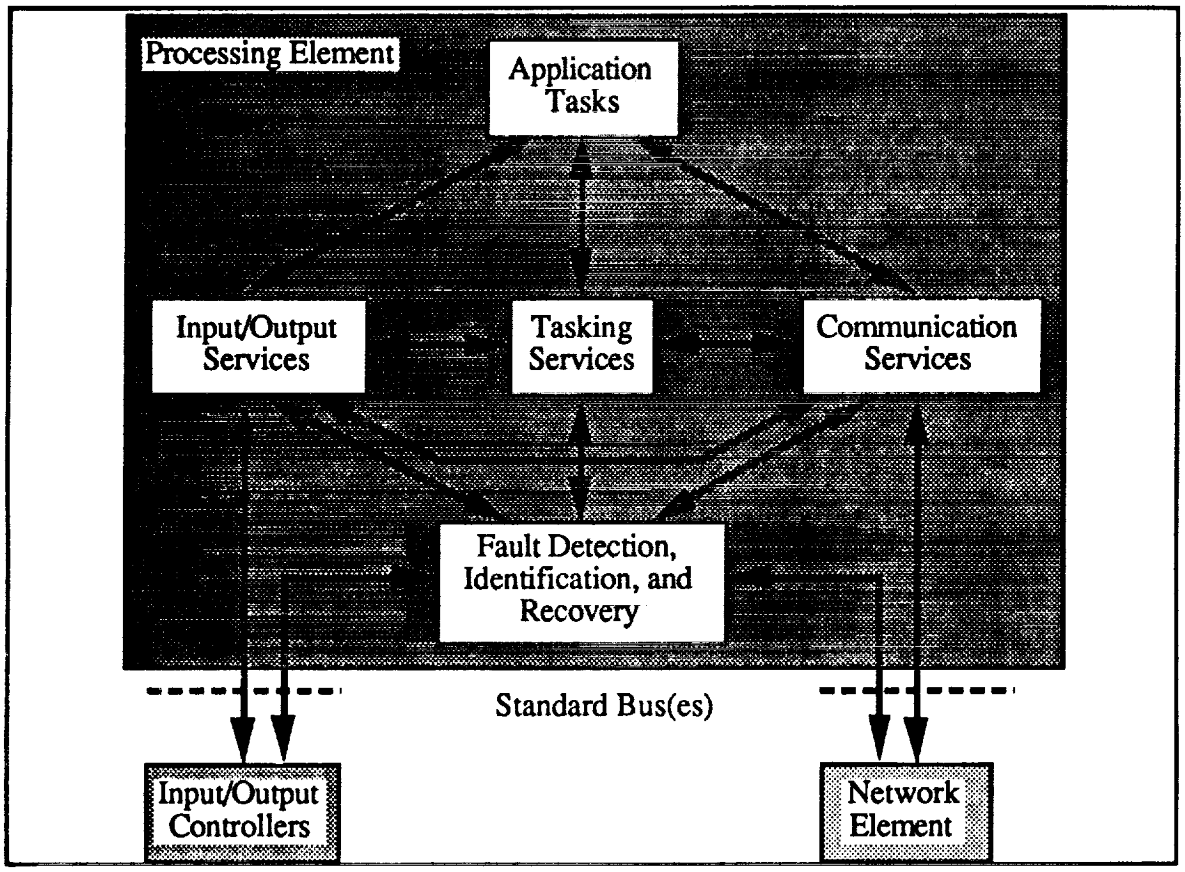


Figure 3-1. AFTA Abstract Structure

The AFTA Communication Services run on the Processing Elements and interface to the Network Elements over a standard backplane bus. The AFTA Input/Output Services also run on the Processing Elements and communicate with the Input/Output Controllers over a standard backplane bus, which may be separate from the bus which hosts the Processing Element-Network Element interface.

3.1. Byzantine Resilience Approach to Fault Tolerance

For a computer to be considered adequately reliable for life- or mission-critical applications, it must be capable of surviving a specified number of component failures with a probability approaching unity. A conservative failure model is to consider failures as consisting of arbitrary behavior on the part of failed components. This type of fault, known as a *Byzantine fault*, may include stopping and then restarting execution at a future time, sending conflicting information to different destinations, and, in short, anything within a failed component's power to attempt to corrupt the system.

Since the concept of Byzantine resilience is central to the theory and operation of the FTPP, it is important to discuss the motivation for this seemingly extreme degree of fault tolerance. Cost-effective validatability and achievement of high reliability are important motivating factors.

Validation-based motivation for Byzantine resilience is perhaps best viewed in the context of an example. We suppose that a digital computer system having a maximum allowable probability of failure of 10^{-9} per hour is required, and that this system must be constructed of replicated channels each of which has an aggregate failure probability of 10^{-4} per hour. Consider a traditional system Failure Modes and Effects Analysis (FMEA)-based approach to achieving the requisite failure rate: likely failure modes of the system are analyzed, their likely extent and effects are predicted, and suitable fault tolerance techniques are developed for each failure mode which is considered to possess a reasonable chance of occurring. For the system to meet the reliability requirement, the probability that any given fault is not covered must be less than $\approx 10^{-9}/10^{-4} = 10^{-5}$; that is, it is necessary that the likelihood of a failure occurring which was not predicted and planned for must be less than $\approx 10^{-5}$. Viewed another way, it is (or should be) incumbent upon the designer to prove to an aggressive and competent inquisitor such as a certification authority that fewer than one in 100,000 faults which could occur in the field (as opposed to those induced or injected in the laboratory) could conceivably defeat the proposed fault tolerance techniques. If this assertion cannot be demonstrated within a reasonable amount of time and money, then it is not feasible to validate the FMEA assumptions and hence the claimed 10^{-9} per hour failure rate.

The FMEA process is tedious, time-consuming, and extremely expensive. This is attested to by the seemingly contradictory trend of increasing costs of digital avionics systems even as the cost of hardware continues to decline. This is at least partially due to the fact that the cost of validating critical systems completely overwhelms the cost of their design and construction. Software validation is a major component of this cost, and inappropriate fault tolerance-related architectural features only aggravate the difficulty.

In contrast consider another fault tolerance technique which guarantees that the system can tolerate faults, without relying upon any *a priori* assumptions about component misbehavior. In effect, a faulty component may misbehave in any manner whatsoever, even to the extreme of displaying seemingly intelligent malicious behavior. A system tolerant of such faults would obviate the expensive and physically intractable problem of convincing a knowledgeable inquisitor of the validity of restrictive hypotheses regarding faulty behavior, in effect permitting faulty behavior to subsume all conceivable FMEAs. Such a system is denoted "Byzantine resilient," that is, capable of tolerating "Byzantine" faults. The source of this terminology may be found in the seminal literature on the theory of ultra-reliable distributed systems [LSP82]:

> "Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement."

The generals in this analogy correspond to Processing Elements in a redundant computing system, the traitors correspond to faulty Processing Elements, and the messengers correspond to interprocessor communications links. It is typically assumed that faulty link (traitorous messenger) behavior is subsumed by faulty source Processing Element (traitorous general) behavior.

One expects a system capable of tolerating such a powerful failure mode to be intrinsically complex and possess numerous inscrutable and exotic characteristics. To the contrary, the requirements levied upon an architecture tolerant of Byzantine faults are relatively straightforward and unambiguous, simply comprising a lower bound on the number of fault containment regions, their connectivity, their synchrony, and the utilization of certain simple information exchange protocols. We assert that a satisfactory demonstration that an architecture possesses these simple attributes is far less expensive and time-consuming than proving that certain uncovered failure modes can occur with a probability of at most 10^{-5}. Existing critical computing systems are typically designed to be triply or quadruply redundant anyhow; meeting the requirements for Byzantine resilience requires a simple rearrangement of the channels and addition of a few interchannel communication protocols. We think this minor rearrangement of the architecture recovers many times over the cost of an FMEA-based validation. Moreover, it is our experience that the run time overhead required to achieve Byzantine resilience can be substantially less than that required to achieve significantly lower levels of fault coverage using fault tolerant techniques based on restrictive hypothetical models of failure behavior.

By making the system Byzantine resilient, in our opinion we have imparted it some powerful programming attributes which result in a significant reduction in software validation effort and cost. First, the hardware redundancy is largely transparent to the programmer. The applications programs and the operating system are developed, debugged, and validated in a simplex (nonredundant) environment without any regard for the redundant copies of the software executing on redundant hardware. Second, the management of hardware redundancy is transparent to the programmer. The applications programs and the operating system are rigorously separated from the hardware and software that manages redundancy. Redundancy management includes functions for detection and isolation of faults, masking of errors resulting from faults, and reconfiguration and reallocation of resources. This rigorous separation allows independent validation of various software entities such as the applications programs, the operating system, and the redundancy management software. By breaking the destructive synergism that comes from intertwining these entities, significant reduction in software validation effort has resulted for the FTPP's predecessors, including the Fault Tolerant MultiProcessor (FTMP) [Lal86a], the Fault Tolerant Processor (FTP) ([Lal86b]), and the Advanced Information Processing System (AIPS) ([Lal84], [Lal85]). Third, a guarantee is made to the applications programmer and the operating system on interprocessor message ordering and validity which holds in the presence of arbitrary faults, and relieves the programmer from consideration of faulty behavior when designing a distributed application. These guarantees are embodied in the Byzantine Resilient Virtual Circuit (BRVC) abstraction of the FTPP and are explained in greater detail below. Once again, the practical impact of this abstraction is the reduction of effort required to validate distributed applications software executing on the FTPP.

It has been suggested that Byzantine resilient systems are overdesigned because such strange failure modes cannot occur in real life. On the contrary, we contend that odd unanticipated failure modes occur often enough in practice that their probability of occurrence cannot be dismissed, and that ultra-reliable computing systems must be able to tolerate them. We present three examples as evidence.

At least one in-flight failure of a triplex digital computer system was traced to an apparent Byzantine fault and the lack of appropriate architectural safeguards against such faults ([Ma78], [Lal86b]). In circuit switched network studies at Draper, a failure mode was observed in which a faulty node responded to commands addressed to any node. The garbled response resulted in identification of innocent nodes as failed, until more sophisticated tests were carried out specifically with this failure mode in mind. A failed processor sending different information to two other processors was observed in the SIFT computer†. Unless appropriate effort and architectural features are employed to survive, categorize, and analyze failures, Byzantine failures are difficult to identify, making it likely that many other undiagnosed cases of Byzantine failures have occurred.

Because such failure modes exist in practice, an ultra-high reliability system must be able to tolerate them. Diagnosis of arbitrary failure behavior requires comparison of the device under test with one having at least as many states [Sun74], implying that component replication is required. The replicated components must be provided with bitwise identical inputs, upon which they perform identical operations. Fault masking, detection, and diagnosis are obtained via bitwise comparison of outputs.

Several key issues arise from the requirement that the system must obtain bitwise consensus on input information such as nonredundant sensor data in the presence of Byzantine failures. Specifically, Byzantine resilient input consensus protocols must be implemented. Theoretical studies have resulted in several prerequisites for protocols which correctly function in the face of arbitrary failure behavior by "f" participants in the protocol. These requirements, appropriate for deterministic and unauthenticated protocols, may be summarized as:

1. There must be at least 3f+1 participants in the protocol [Pe80].
2. Each participant must be connected to each other participant through at least 2f+1 disjoint communication paths [Dol82].
3. The protocol must consist of a minimum of f+1 rounds of communication among the participants [Fis82].
4. The participants must be synchronized to within a known skew of each other [Dol84].

† Personal communication with D. L. Palumbo, NASA Langley Research Center, 1987.

A system which meets these prerequisites is called "f-Byzantine resilient." In a minimal 1-Byzantine resilient processing site, four participants, each of which is connected to each other participant by three disjoint communication paths, must execute a synchronous two-round protocol to obtain consensus in the presence of a Byzantine fault. The FTPP is designed in accordance with these architectural precepts.

3.2. Physical Architecture

The basic unit of the FTPP is the *cluster* (Figure 3-2) which contains at least 4 Network Elements and the associated Processing Elements. The *Network Element* (NE) is a CSDL-designed component, at least 4 of which are fully connected and operate in tight synchrony to perform message exchanges. Messages entering the NEs are exchanged and voted according to the class parameter of the message. For instance, a source congruency message requires 2 rounds of exchange within the Network Element core before the voting process occurs, whereas a voted message requires only a single round of exchange. In addition, since messages are addressed using virtual identifiers, the operation of the NE is contingent upon the system configuration to identify the physical hardware associated with the source and destination addresses.

In an FTPP cluster, each Network Element hosts up to 8 *Processing Elements* (PE) each of which is a standard processor with local memory. Figure 3-2 shows a cluster containing 4 PEs per NE. The prototype laboratory models have employed are Motorola 680x0 processors; however, this selection is not a design criterion and, in fact, the FTPP is capable of supporting heterogeneous processors. Each Processing Element communicates with the hosting NE via transmit and receive ports which the processors access via a standard bus such as VME, VSB, PI-Bus, Futurebus+, etc.

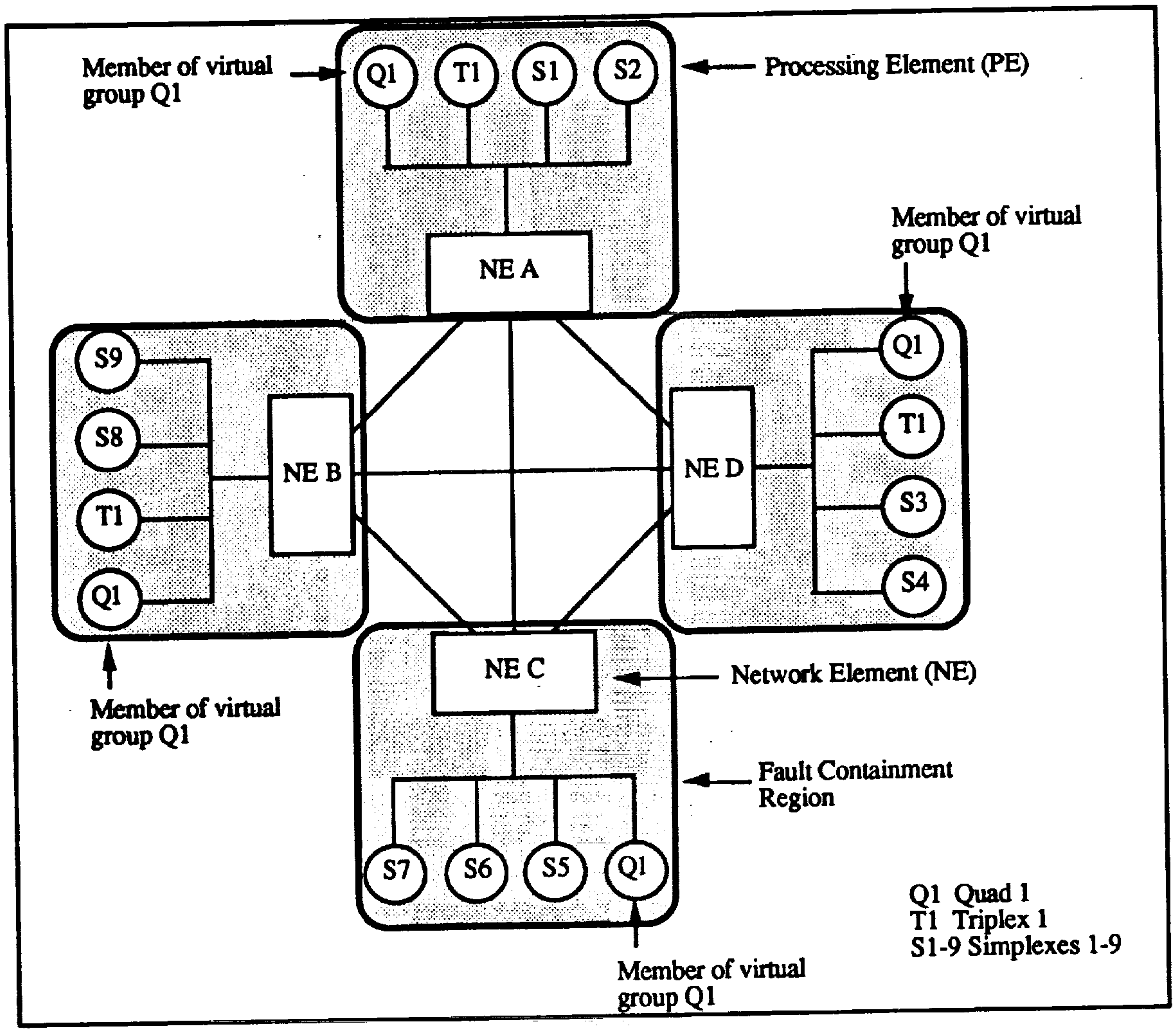


Figure 3-2. FTPP Cluster Architecture

Each Network Element and the associated Processing Elements comprise a *fault containment region* which satisfies the requirements for fault containment, namely, electrical isolation, physical isolation, independent power and independent clocking.

Virtual groups (*VG*) are logical views of the processing resources capable of accepting work in a parallel processing environment. Using this concept, the physical addresses of the Processing Elements as well as the redundancy level of a processing group are concealed from the view of the programmer. A unique identifier is assigned to each Virtual Group; this is the Virtual Group identifier (*VID*).

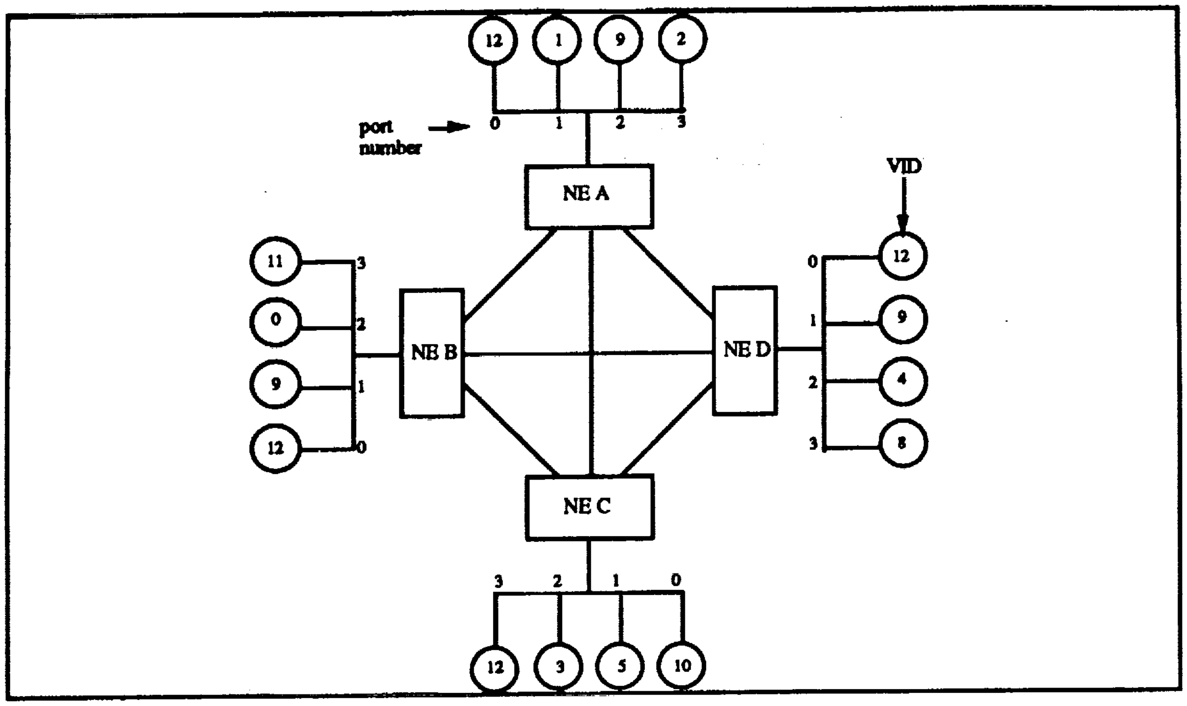


Figure 3-3. FTPP Cluster Architecture - Sample Configuration

Virtual groups are composed of from 1 to 4 Processing Elements; consequently, they may be *simplex*, *triplex* or *quadruplex*. Within a VG, each Processing Element is a *channel* (also referred to as a *member*). When operating redundantly, each processor within a VG executes a suite of tasks which are functionally congruent with the other members of its VG. On the other hand, simplex VGs are merely individual processors executing tasks with no redundancy.

In order to satisfy the theoretical requirements of Byzantine resilience, VGs are comprised of Processing Elements each of which must be resident in a different fault containment region. For example, a quadruplex would comprise Processing Elements resident on each NE.

Fault tolerance on the FTPP is ensured by grouping 3 or 4 Processing Elements into VGs called *fault masking groups* (VG). Fault manifestations in a fault masking group can occur without any degradation in system performance or correctness. Furthermore, these faults can be readily diagnosed.

	VID	NE id	Port number
Q1	12	A	0
		B	0
		C	3
		D	0
T1	9	A	1
		B	2
		D	1
S1	1	A	1
S2	2	A	3
S3	8	D	3
S4	4	D	2
S5	5	C	1
S6	0	B	2
S7	3	C	2
S8	11	B	3
S9	10	C	0

Figure 3-4. FTPP Sample Configuration Table

The system *configuration table* is the mapping of Processing Elements to the Virtual Group identifiers. This mapping identifies the NE hosting the Processing Element as well as the port address through which the Processing Element communicates with the NE. Since all communication within the system is based upon the VID, the system configuration table is resident in the Network Elements as well as in the Processing Elements. Maintenance of this table is provided by a special broadcast message interpreted by both Processing Elements and NEs and by adherence to a strict protocol when the system configuration is modified.

Figures 3-3 and 3-4 illustrate a sample system configuration consisting of 1 quadruplex, 1 triplex, and 9 simplexes.

3.3. Virtual Architecture

A parallel processor is usually characterized by a network that provides interconnection between multiple processing sites. Data is passed between processing sites using a message-passing paradigm. In the FTPP, the ensemble of Network Elements provides a virtual bus topology connecting the processing sites. The virtual bus topology of the FTPP is shown in figure 3-5. This figure shows several example Virtual Groups.

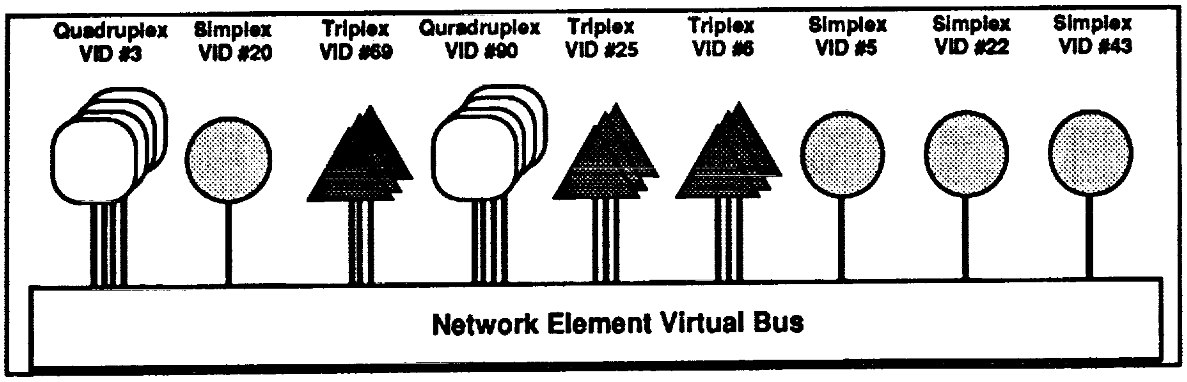


Figure 3-5. FTPP Virtual Configuration

3.4. Communication Mechanisms

Virtual groups communicate via messages which are of 2 basic categories: voted messages and source congruency messages. A *voted message* is sent by all members of a redundant processing group. This message type is employed only when exact consensus amongst all redundant members is expected. Conversely, a *source congruency message* is originated by a simplex Processing Element or by a single member of a redundant processing group requiring a channel-specific exchange of information. Congruency is a generic term which connotes bitwise identical data and computational operations in nonfaulty members of a redundant VG.

Each member of a VG requests a message transmission by sending the message body to its associated transmit port followed by storing the message class in the associated class port. If a majority of members of a VG request transmission, the class is voted by the NE core to determine the exchange and voting mechanisms to employ. In addition, the destination VID is also voted. Subsequently, the message body is handled according to the message *class* which is of 4 basic types: voted messages, source congruency messages, a synchronization message, and configuration update messages.

Message processing within the Network Element core is handled on a VG-by-VG basis. When a majority of the members of the source VG request transmission of a message, that message is eventually processed and delivered to all members of the destination VG.

The ordering of messages to the destination VG is preserved, thereby guaranteeing that all members of the destination VG receive messages in the same order.

Redundant members of VGs execute *functionally congruent* tasks. Functionally congruent tasks are defined to be those which, given identical inputs, may be expected to produce identical outputs in the absence of faults. Since their sequence of tasks are congruent across all members, messages transmitted during their normal executing cycles will necessarily be equivalent as well. Therefore, the message streams emanating from the different members will be identical at least in the message class when no fault exists. This concept is the basis of functional synchronization which is discussed in a subsequent section.

3.4.1. Voted Messages

When the redundant members of a VG transmit a *class 1 message* (voted message) the NEs exchange their copies of the message, create a bitwise voted copy of the message, and compare each copy with the voted copy. This final step generates a vote syndrome which is delivered with the message. Network Elements which host members of the destination VG deliver the message to the appropriate port; otherwise, the NEs discard the message.

As an example, Figures 3-6 through 3-8 depict the transmission of a voted message x from triply redundant Virtual Group T1 to triply redundant Virtual Group T2. In the first phase of the transmission, shown in Figure 3-6, the three members of T1 each deliver their copy of message x to their respective Network Elements. In the second phase, depicted in Figure 3-7, all Network Elements broadcast the message copy received from the Processing Elements in the first phase to all other Network Elements. Note that Network Element B broadcasts a null message (denoted "0" in the figure) because it does not host a member of T1. Subsequent to the second phase of the transmission, all Network Elements (including Network Element B) possess three copies of the message, upon which they perform a bitwise majority vote to obtain the voted message denoted x_{voted}. The null message is not voted. Each Network Element which hosts a member of message recipient Virtual Group T2 (Network Elements A, B, and D in this example) then delivers the voted message to the Processing Elements comprising T2, as shown in Figure 3-8. A Network Element which does not host a member of the destination Virtual Group (Network Element C in this example) discards the voted message.

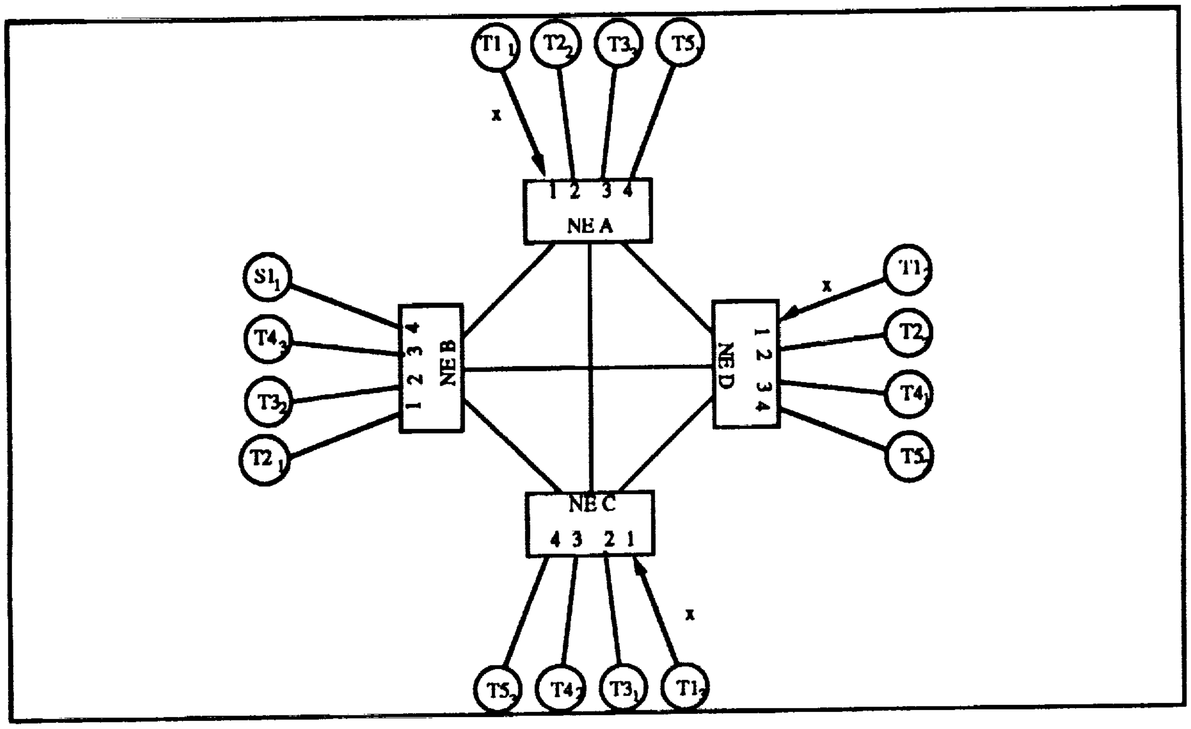


Figure 3-6. Triplex Sender Delivers Class 1 Message x to Network Elements for Transmission

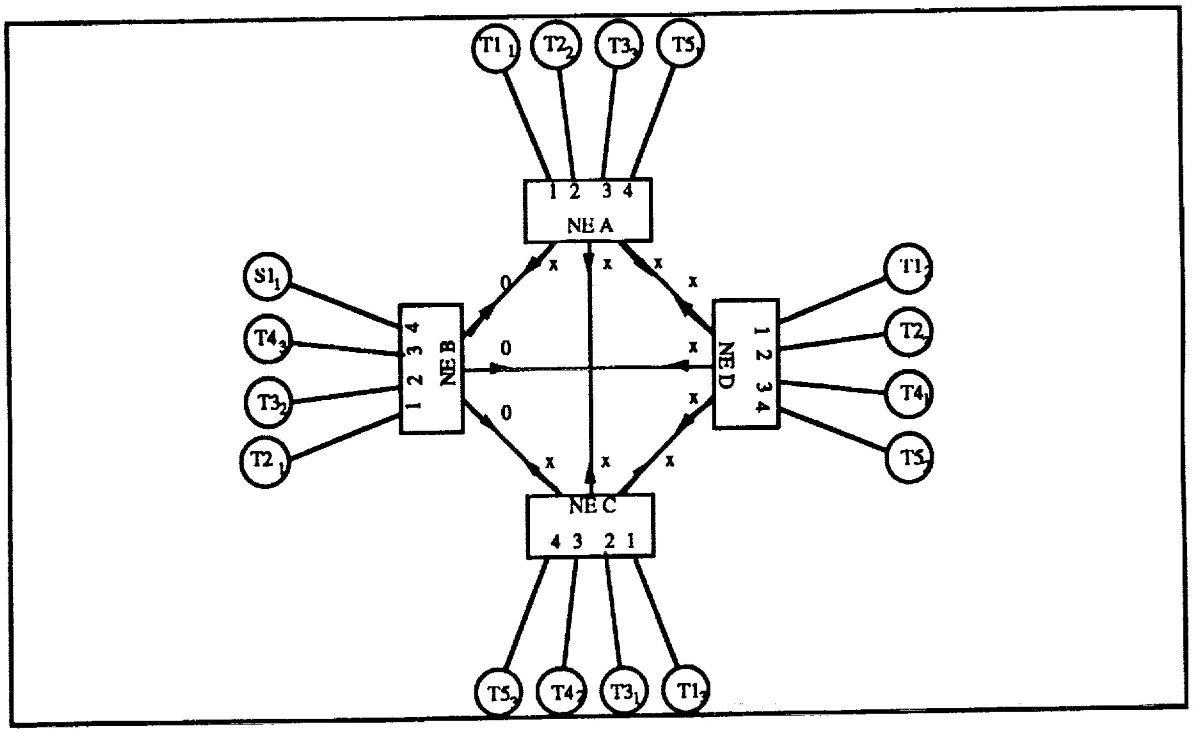


Figure 3-7. Network Elements Perform Mutual Broadcast of Class 1 Message

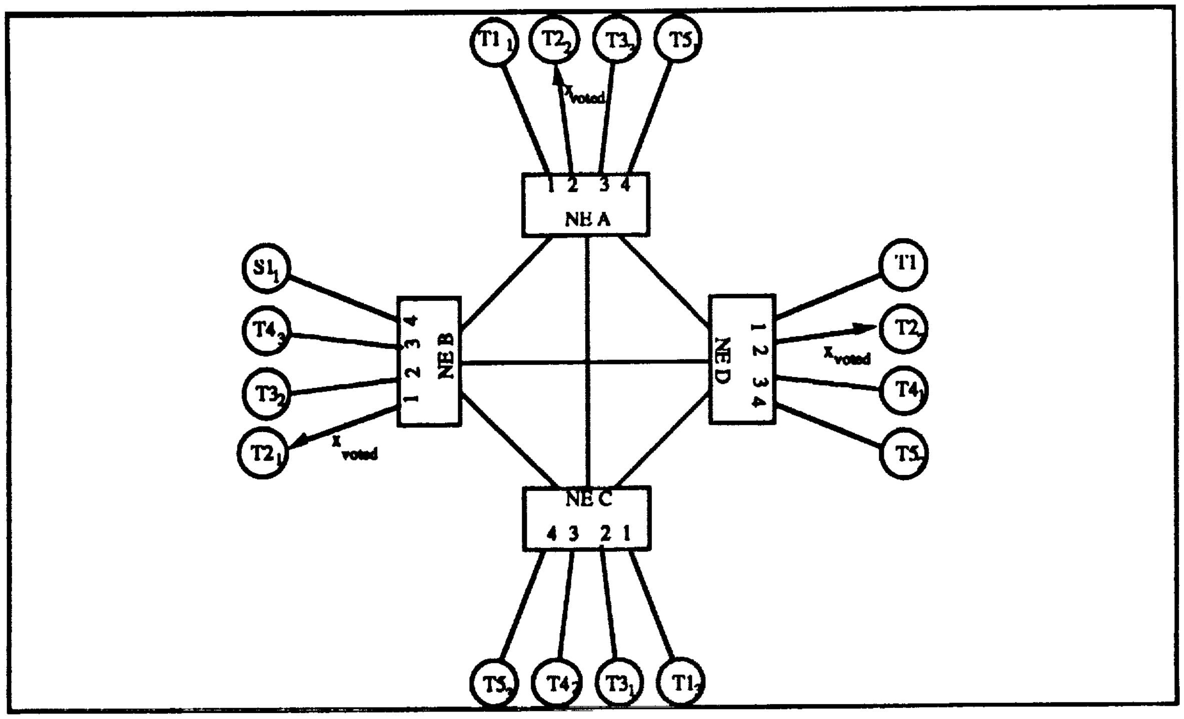


Figure 3-8. Network Elements Vote and Deliver Class 1 Message

3.4.2. Source Congruency Messages

Class 2 messages contain channel-specific information such as the value of a processor clock. The Network Elements perform 2 rounds of exchange of this message, create a bitwise voted copy of the "reflected" copy and compare each "reflected" copy with the voted copy to generate the vote syndrome. Delivery of this message to the destination VG is similar to that of a class 1 message.

Since the NE core can operate on only 1 message at a time, each member of a fault masking group must agree upon which member's channel-specific data are being exchanged. This is achieved by the definition of 5 different class 2 messages. A "class 2 from *x*" message identifies the VG member in FCR *x* (that is, on NE *x* FCR bus) as the source of the message information. However, all members of the VG must participate in the transmission of the "class 2 from *x*" message. This requirement is necessitated by the fact that the NEs vote the message class from each member of the transmitting VG. For a VG with members on NEs A, B and D, if one member transmits a "class 2 from A", the second member sends a "class 2 from B" and the third member sends a "class 2 from D" simultaneously, there would be no consensus on the class of the message. A bitwise voted class is generated and the messages from each member of the sending VG are handled according to this voted class. Therefore, in order to perform an exchange of information where each member receives each other's copy of some information, a series of messages containing this information must be sent by each member of the VG. Each member of a triply redundant VG must sequentially send a series of class 2 messages from each member in order to completely exchange their non-congruent information.

Figures 3-9 through 3-11 depict the transmission of a class 2 message from Simplex Virtual Group S1 to Triplex Virtual Group T2. In the first phase of the transmission, shown in Figure 3-9, S1 transfers the message x to its local Network Element. In the second phase, shown in Figure 3-10, the Network Element hosting S1 broadcasts the single message x to all other Network Elements. The Network Elements which do not host S1 all broadcast null messages, denoted by "0" in the figure. At the end of the broadcast phase, each Network Element has a copy of the message x. Let x_1, x_2, x_3, and x_4 denote the messages received in the first broadcast by Network Elements A, B, C, and D, respectively. (Network Element A is considered to have included itself in the broadcast.) Note that, in the presence of a Byzantine fault on the part of Network Element A, x_1 through x_4 may all be different. Consequently, a second broadcast phase is entered in which all Network Elements broadcast the message received on the first round, as shown in Figure 3-11. This phase is often denoted the "reflection" phase. After this second broadcast, all Network Elements have a set of four messages which may be bitwise voted to derive x_{voted}. The delivery of the example class 2 message is identical to the delivery of a class 1 message depicted in Figure 3-8. Each Network Element which hosts a member of message recipient Virtual Group T2 (Network Elements A, B, and D in this example) delivers the voted message to the Processing Elements comprising T2, as shown in Figure 3-8. Again, a Network Element which does not host a member of the destination Virtual Group (Network Element C in this example) discards the voted message.

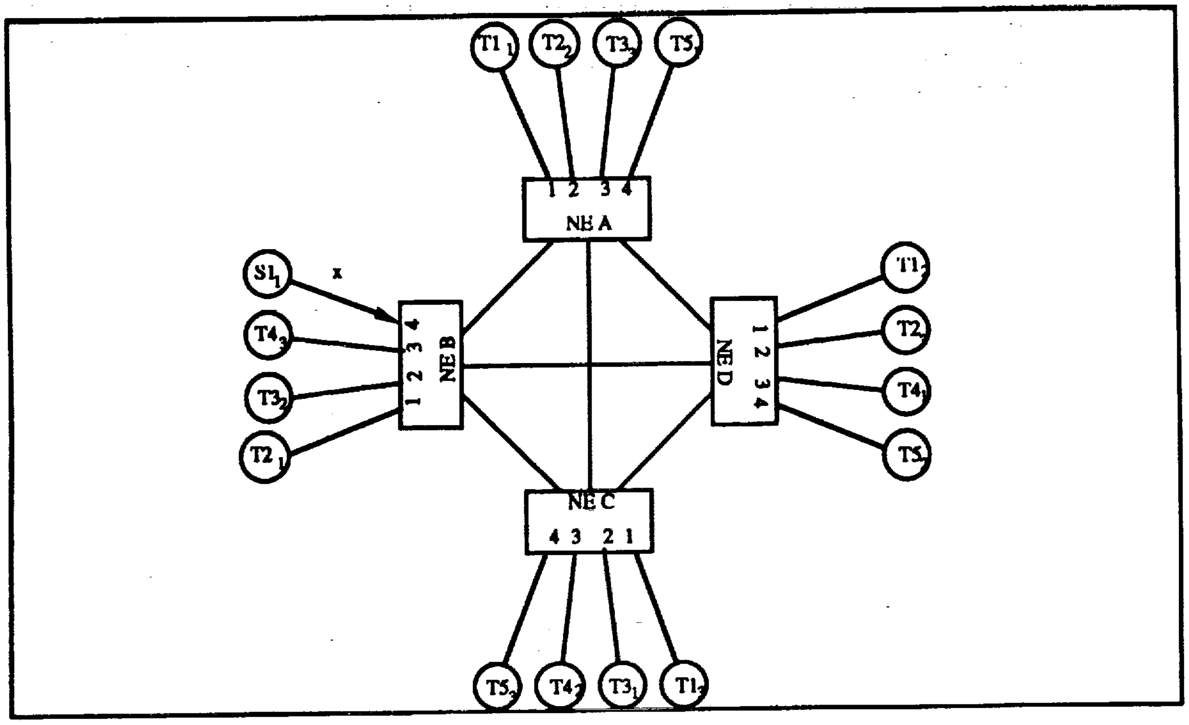


Figure 3-9. Simplex Sender Delivers Class 2 Message x to Network Elements for Transmission

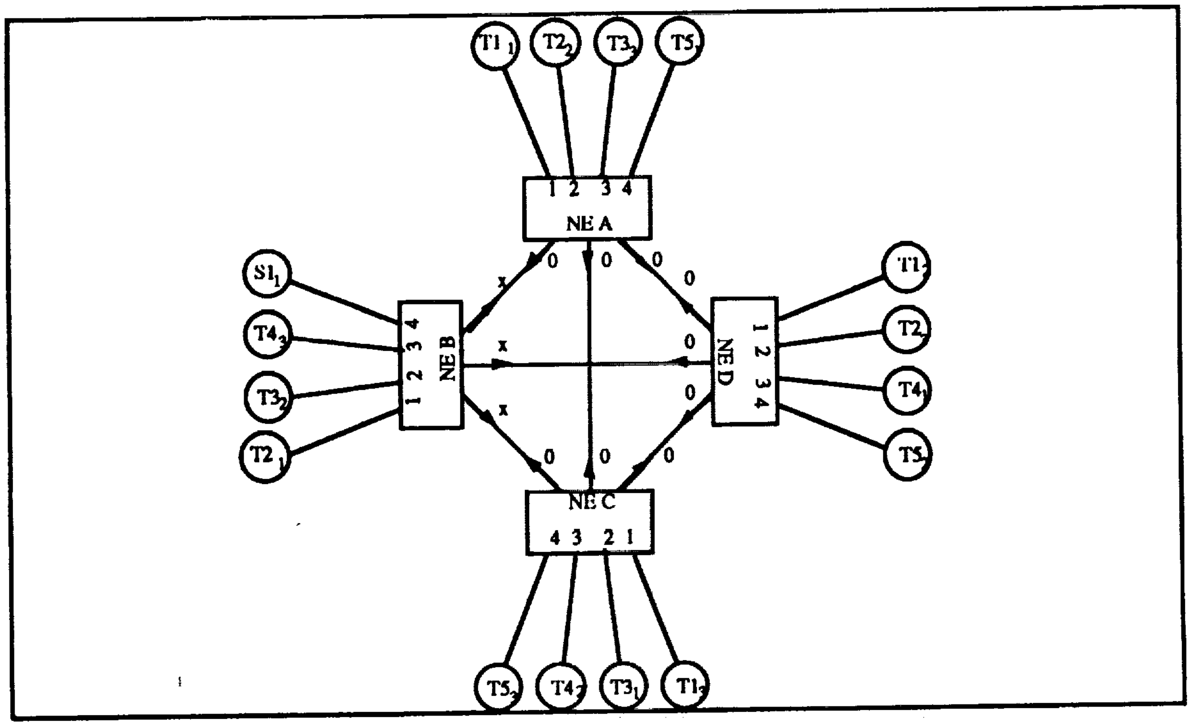


Figure 3-10. Network Element B Performs Initial Broadcast of Class 2 Message

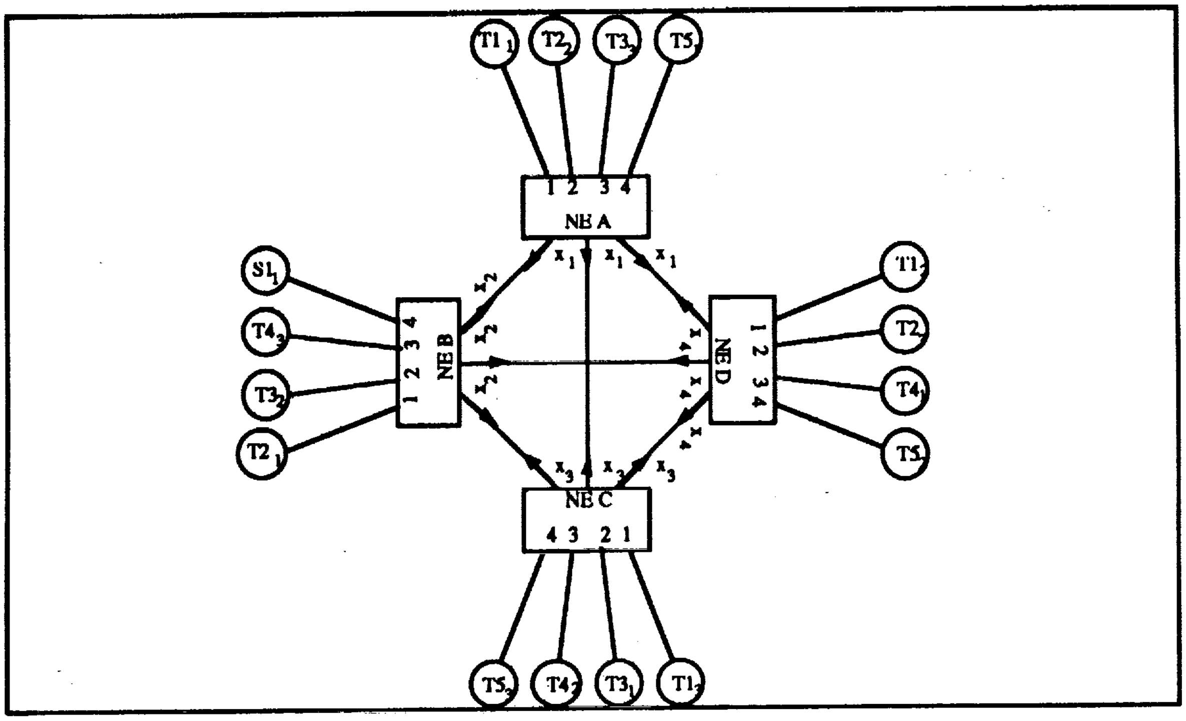


Figure 3-11. Network Elements Perform Mutual Broadcast of "Reflected" Class 2 Message

3.4.3. Synchronization Message

The synchronization message is a special message type which contains no message body. It is employed as an expeditious synchronization mechanism among members of a VG.

3.4.4. Configuration Update Message

Since the NE core is cognizant of the mapping of VGs to physical hardware, a special configuration update message is defined. This message is composed by a VG in a specific format such that it can be intercepted by the NE core in transit to all VGs. The configuration update is effective immediately.

3.4.5. Synchronization

The FTPP uses a scheme denoted Functional Synchronization to obtain synchronization of processors composing a Virtual Group. In functional synchronization, events are equated to the occurrence of unambiguously defined actions performed by the members of a Virtual Group during the course of their normal task execution. For example, sending a message could be defined to constitute an event, as could reading an incoming message buffer or rescheduling a process. A frame is defined by the occurrence of a specified number of these events. When the requisite number of events has transpired, the VG executes a *synchronizing act*, which defines the end of the old frame and the beginning of the new. The purpose of functional synchronization is simply to reduce the skew existing between VGs after a synchronizing act to a much smaller value than that existing before the synchronizing act.

As an example of this idea, Figure 3-12 shows the transmission and reception of a message as constituting an event, with one event constituting each frame. However, in functional synchronization, the number of events per frame need not be constant, and can be a function of the length of the current frame (i.e., the time since the last synchronizing act) or any other operational parameter. In addition, the frames of different VGs have no temporal relationship to each other. They may overlap, be of different lengths and periodicities, and exhibit a high degree of disparity.

Three requirements are placed upon members of a VG participating in functional synchronization. First, the members of a VG must exhibit a differential execution rate which can be upper-bounded. A certain degree of nondeterminism and heterogeneity is allowed among the members of a VG, but the maximum difference between the execution rates of the fastest and slowest members of a VG must be known. Second, the length of time between synchronizing acts must also be upper-bounded. A VG cannot continue forever without performing a synchronizing act. The assertion that a VG meets the first two requirements implies that it is possible to upper-bound the time differential with which different VG members arrive at a synchronizing act. Denote this maximum differential or "skew" by σ (Figure 3-12). Members that do not arrive at a synchronizing act within σ time units of a non-faulty member can be assumed to be faulty. The third requirement is that the members of the VG must perform corresponding synchronizing acts in an identical order. An additional requirement, not necessary to obtain functional synchronization but necessary to obtain high coverage fault detection and masking via voting, is that a specifiable subset of messages emanating from the members of a VG must have the characteristic that bitwise agreement on message content implies non-faulty behavior, and bitwise disagreement implies otherwise.

Assume that the members of a VG satisfy the conditions enumerated above. Each member is further assumed to be executing a functionally congruent instantiation of a process which can be unambiguously partitioned into segments. The partitioning may be either manual (the programmer uses a "perform synchronizing act" primitive) or automatic (the compiler or operating system inserts synchronizing acts upon message transmissions, Remote Procedure Call invocations, reading of an input buffer, etc.). Let traversal of a segment boundary constitute an event. Upon the occurrence of such an event, the copy of the operating system resident on each member of the VG decides whether the occurrence of this event is to constitute a frame boundary. As mentioned above, this decision may be a function of the time since the last synchronizing act, the mission phase, or other operational parameters. The only critical requirement is that all copies of the operating system come to an identical conclusion about the status of the event. If the event is determined not to trigger synchronization, the operating system returns control to the process, or, alternatively, to another process awaiting scheduling. Again, all that matters is that these decisions are the same in all members of the VG.

If the operating system instantiations determine that the event constitutes a synchronizing act, they trigger synchronization of the VG hosting the segment instantiations by transmitting a message to themselves into the Network Element core and subsequently awaiting the reception of that message. The difference in time between the delivery of copies of a given message at all non-faulty Network Elements is very small. Therefore the time differential with which the copies of the operating system perceive the arrival of the message of interest is very small (denoted δ in Figure 3-12). Assuming that the operating systems are awaiting this message, they will be synchronized upon its reception.

The Network Element core is a tightly synchronous, clock deterministic aggregate which executes Byzantine Resilient clocking and consensus algorithms in dedicated hardware and firmware. Regarding the present discussion, the Network Element core achieves near-simultaneous delivery of the identical copies of a given message by delaying the transmission and delivery of the message until the *valid message condition* is met by the members of the VG. Satisfaction of the valid message condition depends on the redundancy level of the Virtual Group which sourced the message. For a Virtual Group having triplex or greater redundancy, the valid message condition is fulfilled when all members of a group have requested transmission of a message, or a majority of the members of a group have requested a message transmission and the maximum allowable skew between the transmission of messages by non-faulty members, σ, has expired. For a simplex group, a message is transmitted as soon as the sole member of the group makes the request.

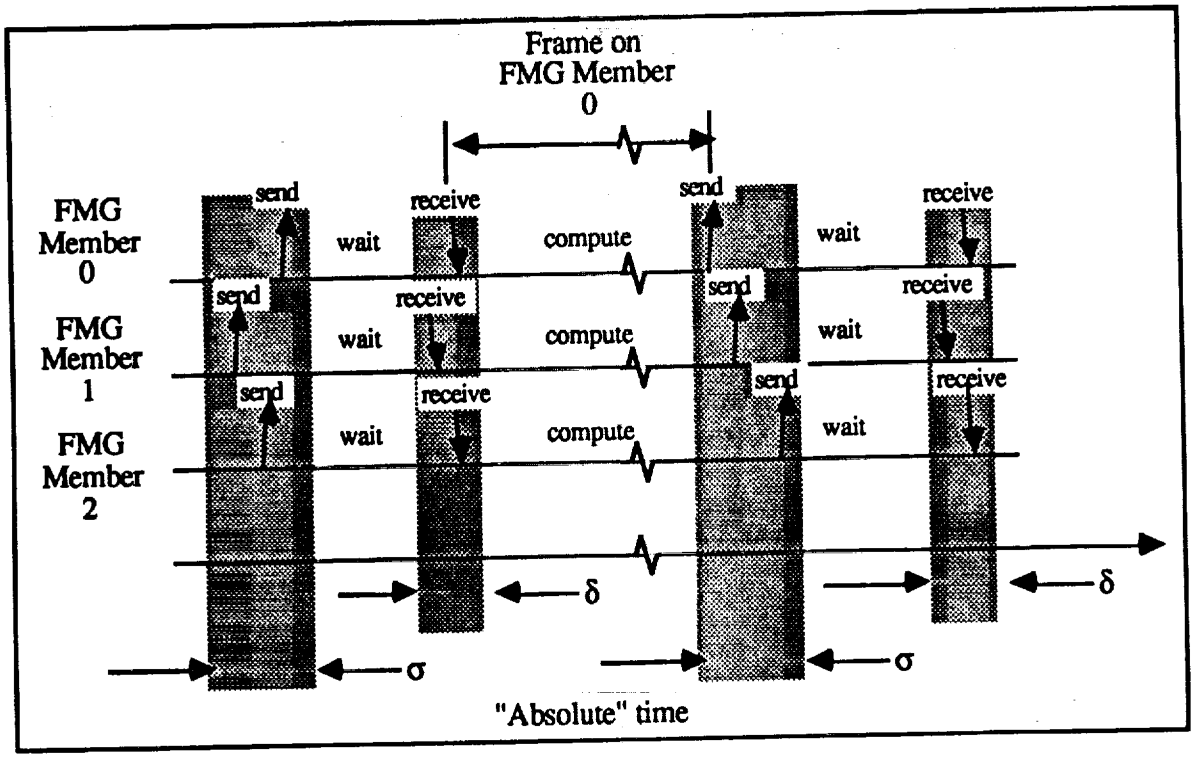


Figure 3-12. Functional Synchronization

The valid message condition is synchronously evaluated for all Virtual Groups by all of the Network Elements in a cluster, using message transmission request patterns that are obtained via a Byzantine Resilient interactive consistency protocol and which are therefore identical at all non-faulty Network Elements. Therefore all Network Elements come to identical decisions about which groups pass the valid message condition. Also by virtue of the tight synchrony of the Network Element core, the message transmission and delivery which may denote the completion of a VG's synchronizing act is performed at very close to the same time in all Network Elements. Thus, VG members awaiting the completion of this transmission perceive the completion at very close to the same point in time, and continue with their execution with a temporal skew which has been reduced by the act of waiting for the completion of the synchronizing act.

No *a priori* relationship is required between the frames of different VGs. This allows a total decoupling of the operational characteristics of different VGs, while nevertheless allowing them to be integrated into a single physical and logical entity in a conceptually coherent manner. Functional synchronization is conceptually transparent to the programmer, who has the option of never knowing that certain acts like reading input message buffers are events or synchronizing acts. In a distributed algorithm, the sending and receiving of messages is likely to be frequent enough to obtain ample opportunity for synchronizing acts. This is of course dependent on the application.

In AFTA, functional synchronization is implemented by the operating system by sending a synchronization message every minor frame. In order to reduce skew among the members the operating system blocks awaiting return of the synchronization message. This is discussed further in section 5. Inter-Virtual Group synchronization among VGs is provided simply by sending messages. Each member of the sending VG transmits a message. The Network Element core distributes, votes, and delivers the message to the destination VG as described above.

3.4.6. Byzantine Resilient Virtual Circuit Abstraction

It is important to unambiguously express to the users of an architecture the fault tolerance properties that it embodies. The message passing properties of the FTPP can be concisely expressed in an abstraction called the Byzantine Resilient Virtual Circuit (BRVC) abstraction. This is a guarantee made to the applications programmer on interprocessor message ordering and validity which holds in the presence of Byzantine faults, and relieves the programmer from consideration of faulty behavior when designing a distributed application. An instance of this abstraction is depicted in Figure 3-13, which shows two loosely synchronized triplex Virtual Groups T1 and T2, each of which contains a faulty member. T1 and T2 are sending messages *a*, *c*, *x*, and *z* to two other triplexes, T3 and T4, which may also contain faulty members. In cases such as these the BRVC abstraction provides the following guarantees:

1. Messages sent by non-faulty members of a source triplex are correctly delivered to the non-faulty members of recipient triplexes.

2. Non-faulty members of recipient triplexes receive messages in the order sent by the non-faulty members of the source triplex.

3. Non-faulty members of recipient triplexes receive messages in identical order.

4. The absolute times of arrival of corresponding messages at the members of recipient triplexes differ by a known upper bound (δ in Figure 3-13).

5. The time between a valid message transmission request and message delivery possesses a known upper bound (τ in Figure 3-13).

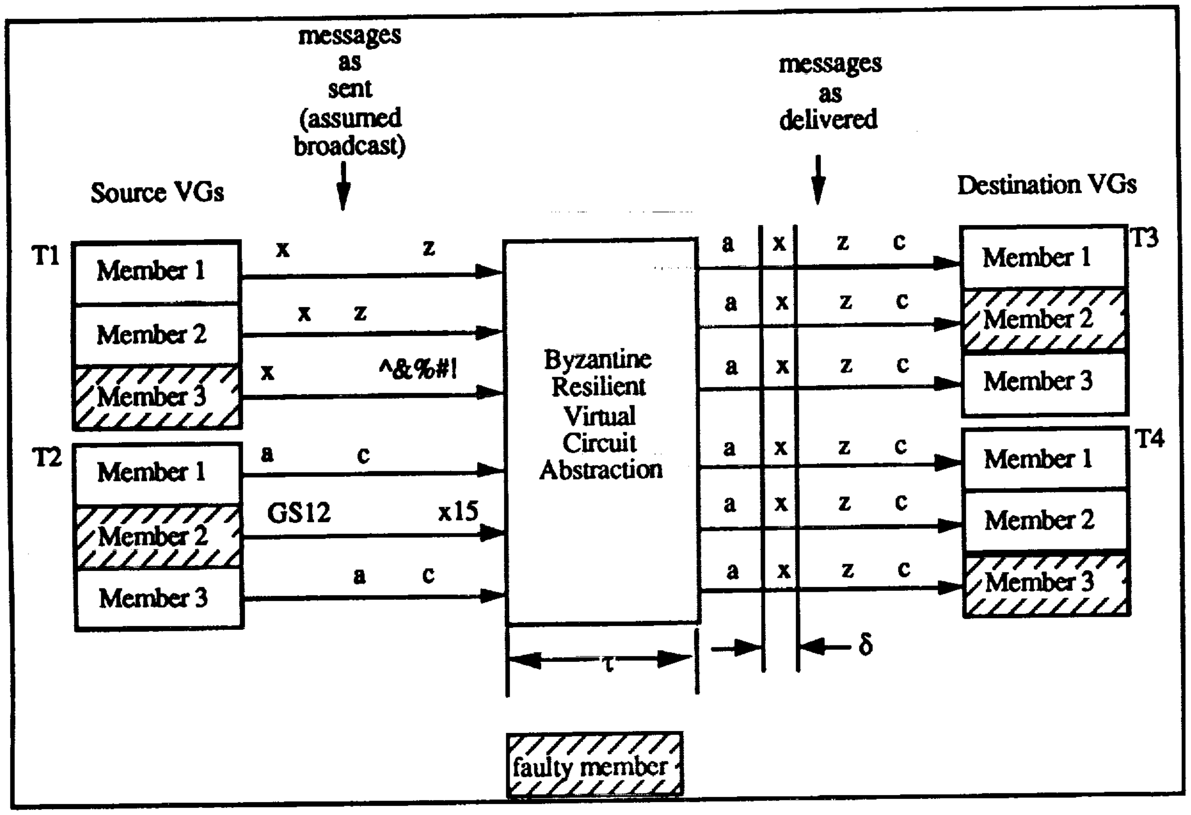


Figure 3-13. Byzantine Resilient Virtual Circuit Abstraction

These guarantees on totally ordered and timely delivery are not typically made by parallel processors even in the absence of faults.

3.4.7. Scooping

Using the BRVC abstraction and the idea of functional synchronization, a VG can perform a useful act called a *scoop*. A VG may be triggered to perform a scoop by a resident process' request to update its input message buffer. It is critical that the message buffers of all VG members possess identical contents to prevent divergence of any computation resulting from decisions based on those contents. Scooping uses the BRVC abstraction and functional synchronization to obtain a consistently ordered and identical set of messages at each VG member.

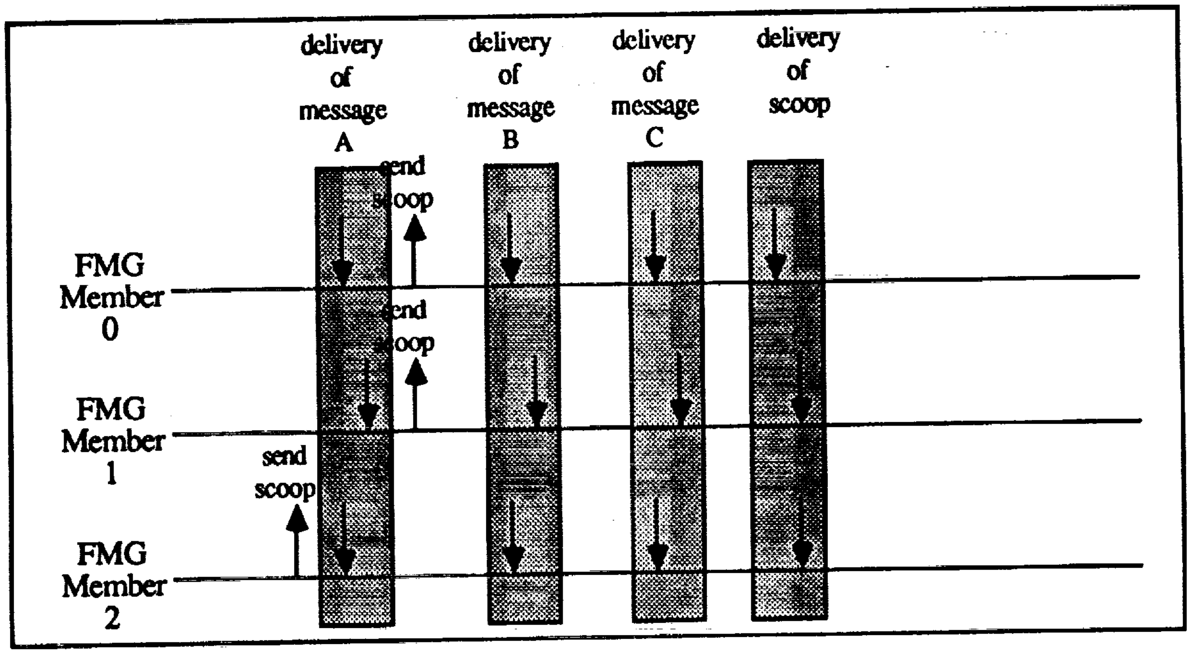


Figure 3-14. Scooping a Message

A VG performing a scoop takes advantage of BRVC by simply sending a message to itself and awaiting its reception. By BRVC guarantee 3, all members receive identically ordered identical copies of each message. By guarantee 4, the absolute times of arrival of each message, and in particular the scoop message, at the different VG members differ by a known upper bound δ. Using these guarantees the recipient VG is assured that each member of the VG has received an identical set of identically ordered messages before delivery of its own scoop message. Therefore any decision made based on messages received prior to a scoop reception will be congruent in all members. Upon delivery of the scoop shown in Figure 3-14, each VG possesses identical input message buffers {A, B, C}. In addition, the VG has executed a synchronizing act.

3.4.8. Input/Output Redundancy Management

The AFTA will interface to a wide variety of inputs and outputs. The objective of this section is to describe the Input/Output process in the AFTA.

A given I/O process may select from among a number of I/O redundancy management options, depending on the redundancy level of the input source, the redundancy level of the VG which executes the Input/Output Request, the redundancy level of the source of the output data, and the redundancy level of the VG for which the input data is destined. It is expected that a given implementation of AFTA will utilize most if not all of these redundancy management options. Moreover, it is expected that several different I/O operations will be in progress at a given time. All I/O procedures depicted below are performed transparently to the application programmer by the AFTA I/O System Services.

A representative set of I/O redundancy management policies has been defined under the Conceptual Study. Each policy will now be described. To allow relatively compact illustrations, a representative cluster containing four NEs and four PEs per NE will be used to demonstrate the I/O processes. In addition, it is assumed that the VGs communicate with the IOCs over the FCR backplane bus. In reality, I/O buses may be separated from the FCR backplane bus for I/O bandwidth enhancement.

3.4.8.1. Input Procedures

3.4.8.1.1. *Case 1: Dumb IOC, Simplex Source in same FCR as IOC, Simplex Destination (same as source)*

Simplex VGs may acquire input from an IOC resident in the same FCR. Neither the operations performed by the simplex VG nor the IOC should be viewed as possessing a significant degree of fault tolerance, so this operation would not be used for critical input acquisition.

Procedure

Step 1: Simplex source (VG 1 in Figure 3-15) accesses the I/O device in the same FCR (IOC1, connected to NE0) to request the input.

Step 2: IOC1 provides the data to S1 over the FCR backplane bus. S1 can now use the input data.

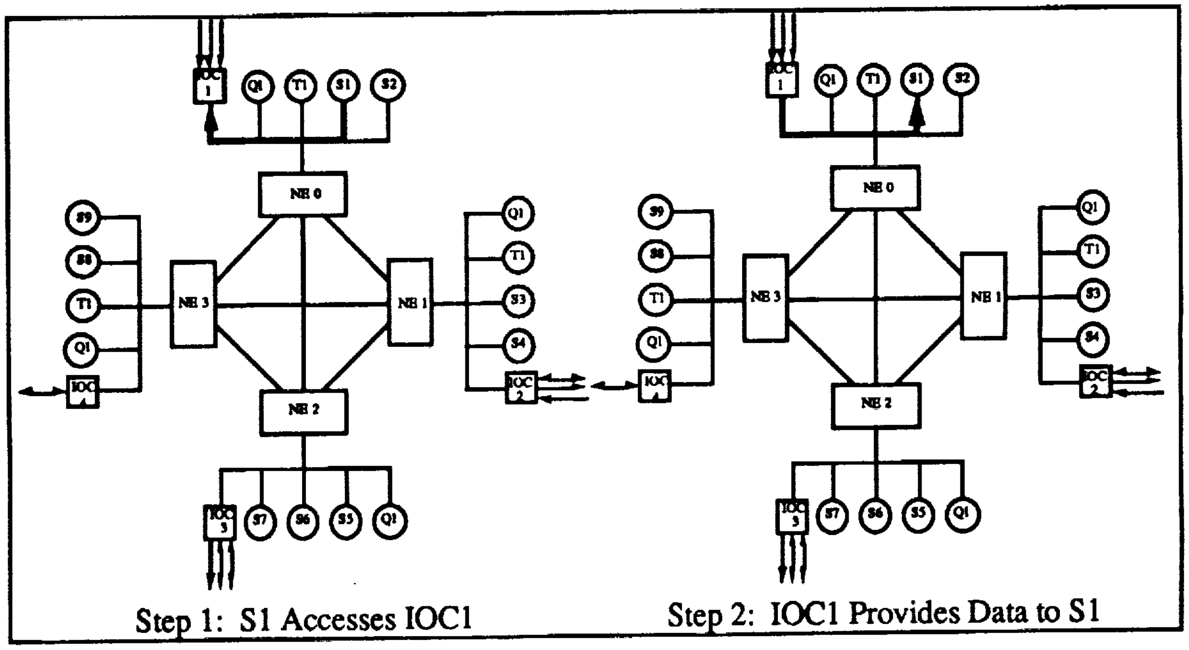


Figure 3-15. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data

3.4.8.1.2. *Case 2: Dumb IOC, Simplex Source in same FCR as IOC, Simplex Destination (different from source).*

A simplex VG may wish to acquire input data from an IOC coresident in its FCR and then transmit the possibly processed input data to another simplex in the AFTA. Since only simplex sources and destinations are involved, this procedure is not considered to be fault tolerant.

Procedure

Step 1: Simplex source (VG 1 in Figure 3-16) accesses the I/O device in the same FCR (IOC1, connected to NE0) to request the input.

Step 2: IOC1 provides the data to S1 over the FCR backplane bus.

Step 3: S1 performs Class 2 Exchange (refer to Section 3.4 for description of Class 2 Exchange) to deliver data to destination, say S9. S9 may now use the data.

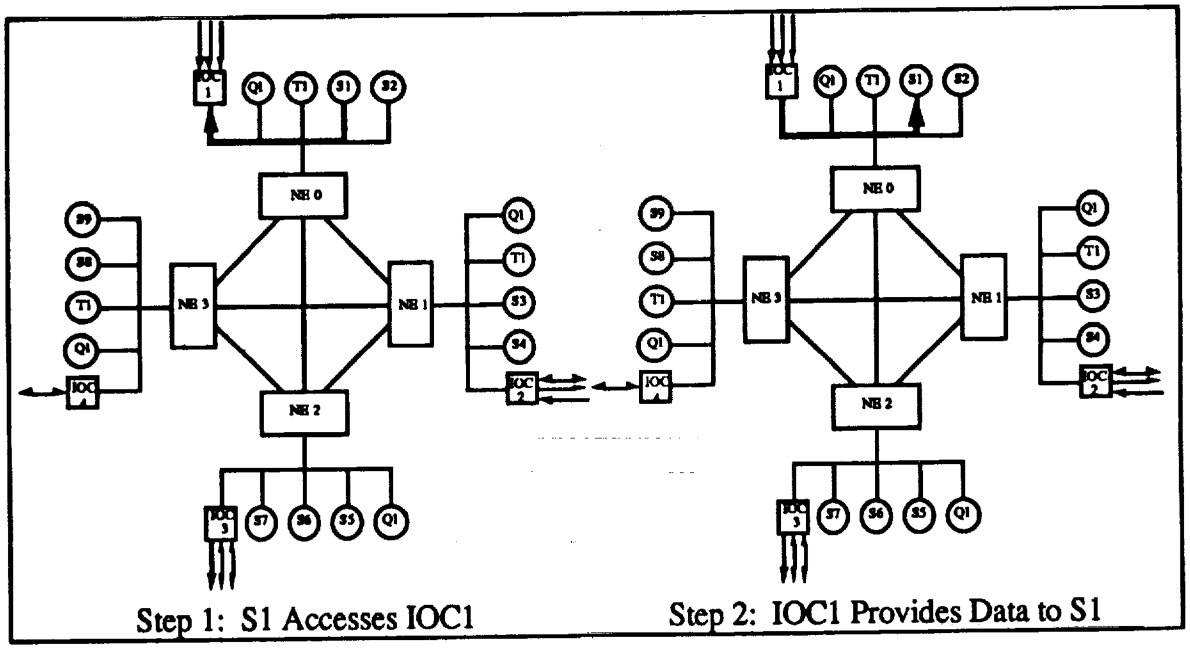


Figure 3-16. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data

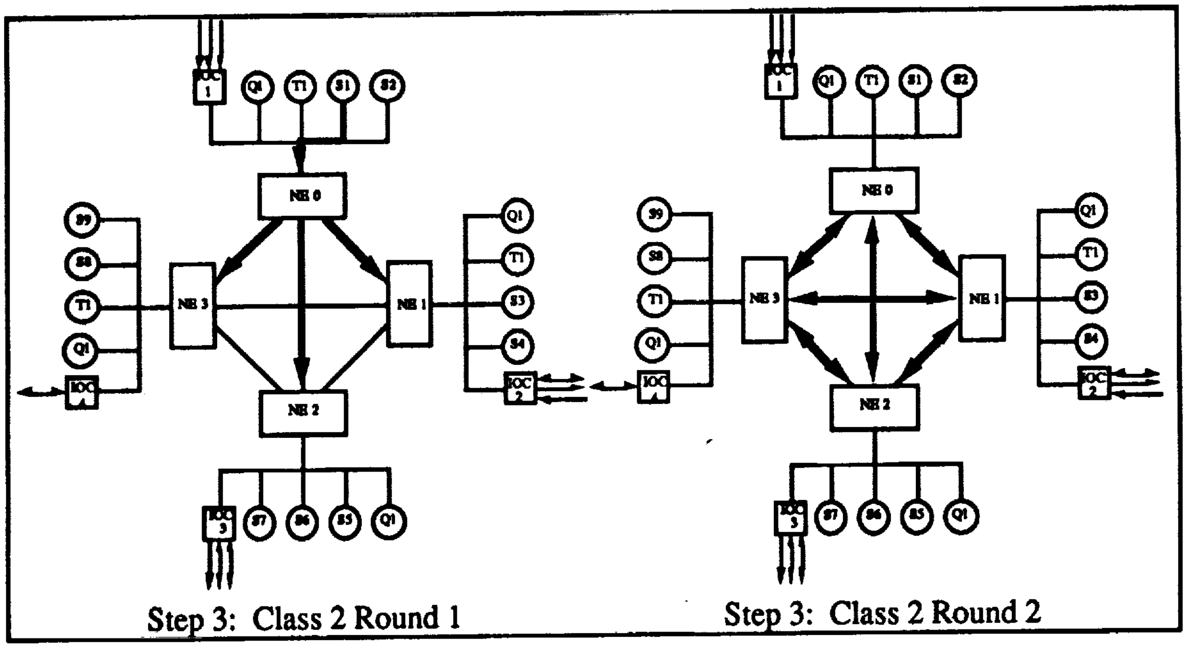


Figure 3-17. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2

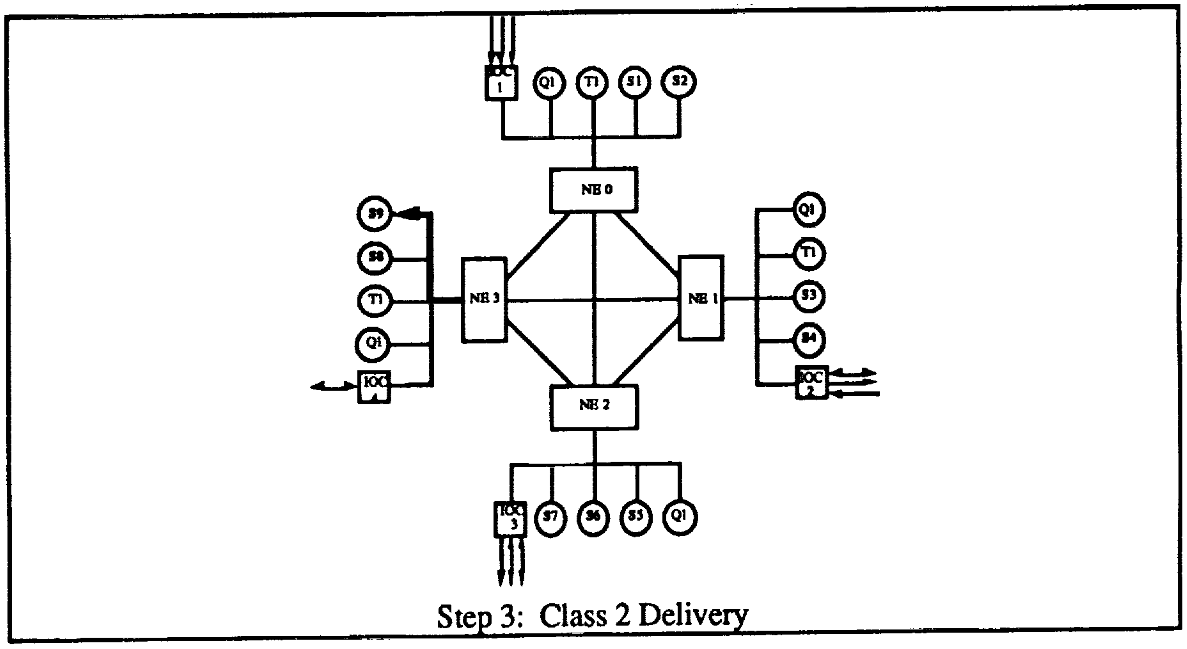


Figure 3-18. Step 3 - S1 Performs Class 2 Exchange, Delivery of Input Data to S9

3.4.8.1.3. Case 3: Dumb IOC, Simplex Source in same FCR as IOC, Triplex Destination

This case comprises the procedure for unreliably reading and possibly processing input data by a simplex VG and coresident IOC, followed by reliably distributing the data to a triplex destination VG. The procedure described below guarantees that each copy of the recipient triplex VG will receive identical (but possibly erroneous) input data.

Procedure

Step 1: Simplex source (VG 1 in Figure 3-19) accesses the I/O device in the same FCR (IOC1, connected to NE0) to request the input.

Step 2: IOC1 provides the data to S1 over the FCR backplane bus.

Step 3: S1 performs Class 2 Exchange (refer to Section 3.4 for description of Class 2 Exchange) to deliver data to destination, say T1. T1 may now use the data.

Note the similarity of Step 3 of Case 3 to Step 3 of Case 2. In general, S1 must perform a Class 2 Exchange to deliver the data to any other VG in the cluster.

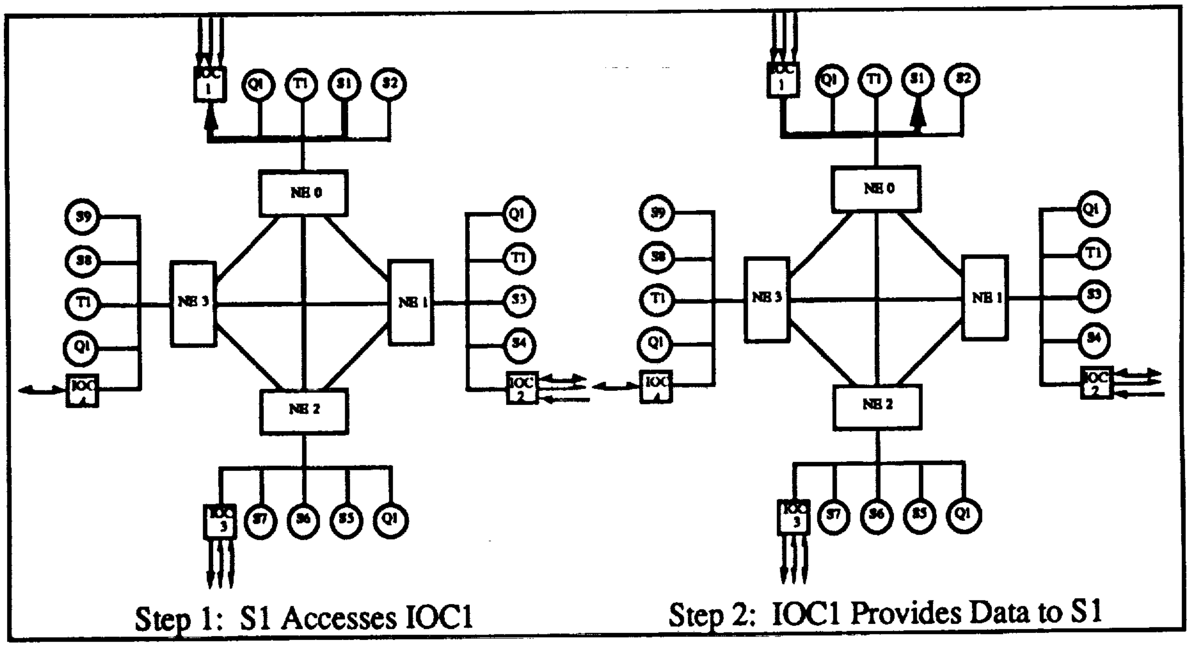


Figure 3-19. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data

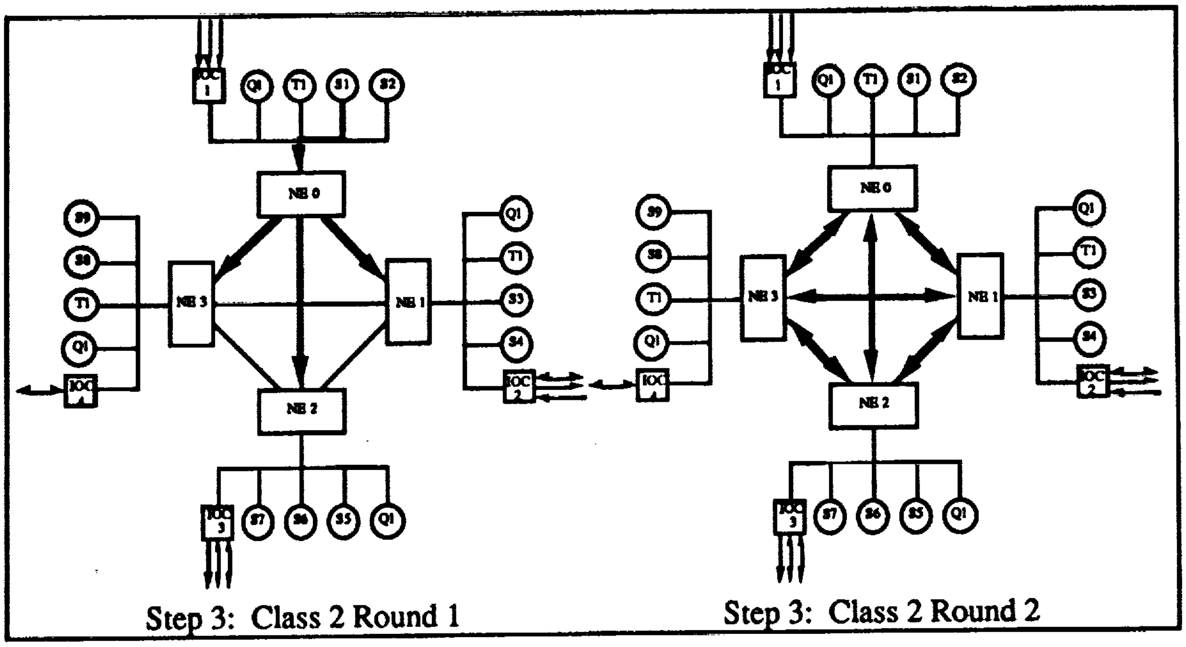


Figure 3-20. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2

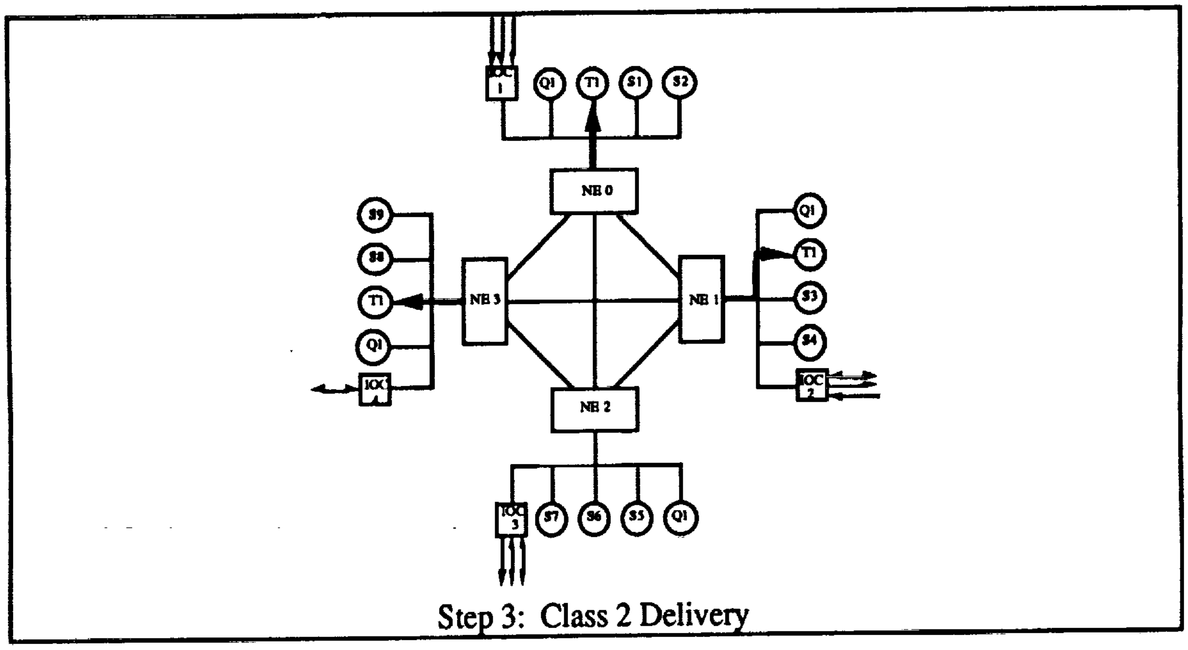


Figure 3-21. Step 3 - S1 Performs Class 2 Exchange, Delivery of Input Data to T1

3.4.8.1.4. Case 4: Dumb IOC, Simplex Source in same FCR as IOC, Broadcast Destination

This case comprises the procedure for unreliably reading and possibly processing input data by a simplex VG and coresident IOC, followed by reliably distributing the data to all VGs in the AFTA. As in the previous case, the procedure described below guarantees that each recipient VG will receive identical (but possibly erroneous) input data.

Procedure

Step 1: Simplex source (VG 1 in Figure 3-22) accesses the I/O device in the same FCR (IOC1, connected to NE0) to request the input.

Step 2: IOC1 provides the data to S1 over the FCR backplane bus.

Step 3: S1 performs Class 2 Exchange (refer to Section 3.4 for description of Class 2 Exchange) to deliver data to all destinations.

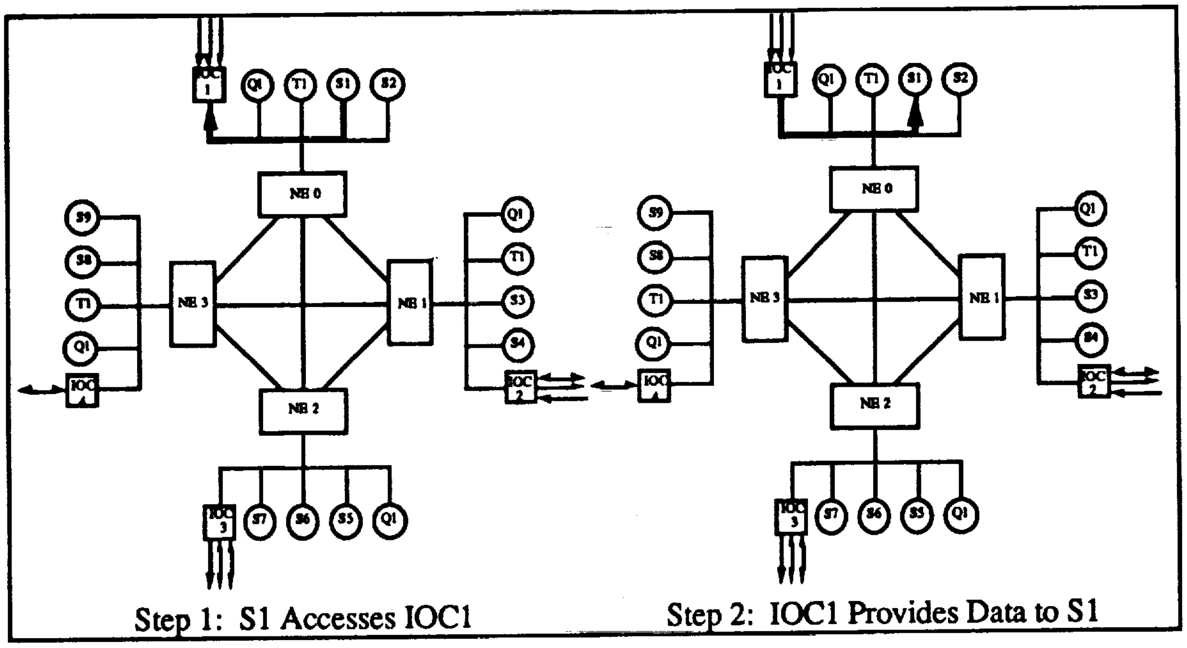


Figure 3-22. Steps 1 and 2 - S1 Accesses IOC1 to Read Input Data

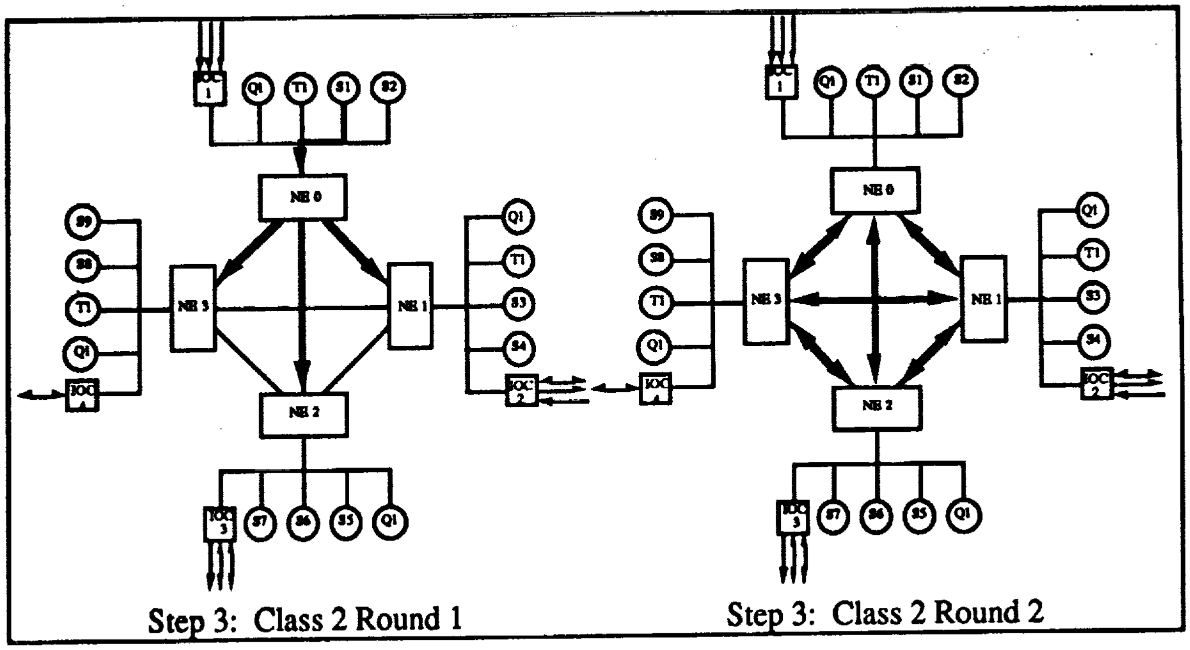


Figure 3-23. Step 3 - S1 Performs Class 2 Exchange, Rounds 1 and 2

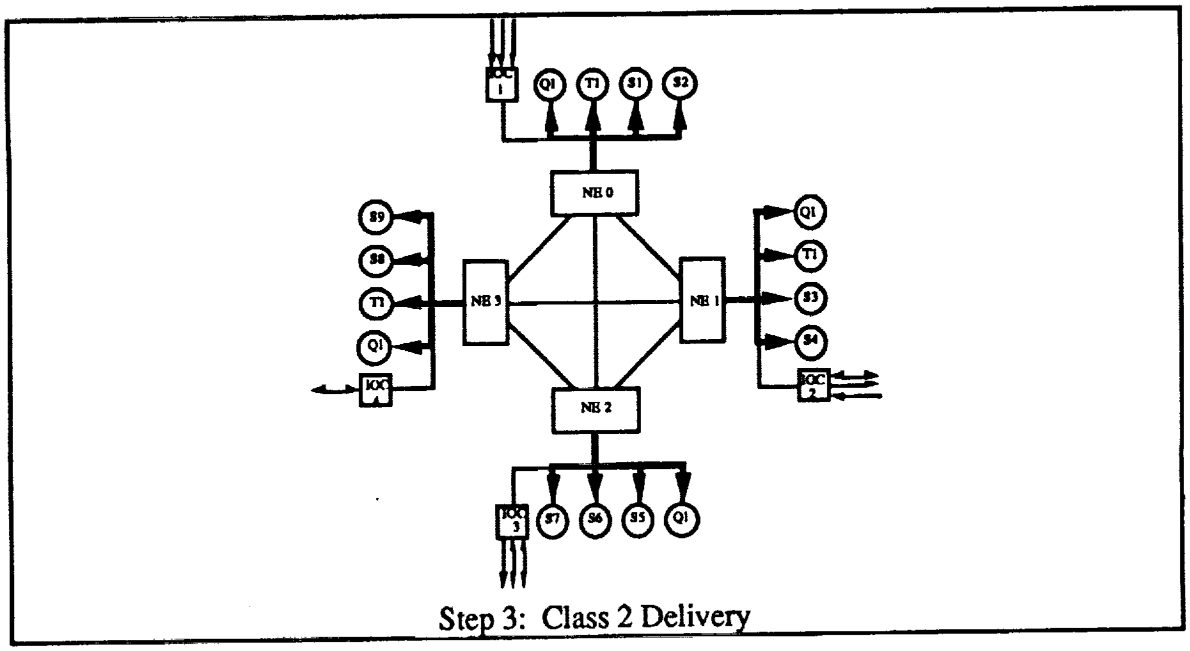


Figure 3-24. Step 3 - S1 Performs Class 2 Exchange to Deliver Input Data to All VGs

3.4.8.1.5. Case 5: Dumb IOC, Triplex Source in same FCR as IOC, Triplex Destination

In this case, one member of a triplex VG is coresident with an IOC which possesses an input data that must be acquired and reliably distributed to all members of that VG. The following procedure will guarantee that all members of the triplex VG will receive identical copies of the input data. Note that since the IOC, its coresident member of the triplex VG, or the NE to which they are attached may be faulty, the nonfaulty recipients may receive identical but erroneous input data.

Procedure

Step 1: Triplex source accesses the I/O device in the same FCR (IOC1, connected to NE0) to request the input.

Step 2: IOC1 provides the data to T1 in NE0 over the FCR backplane bus.

Step 3: T1 performs Class 2 Exchange (refer to Section 3.4 for description of Class 2 Exchange) to deliver data to destination, say T1. T1 may now use the data. Note that any other destination may be selected by T1 for the data, including all VGs in the AFTA (via the broadcast packet exchange mode).

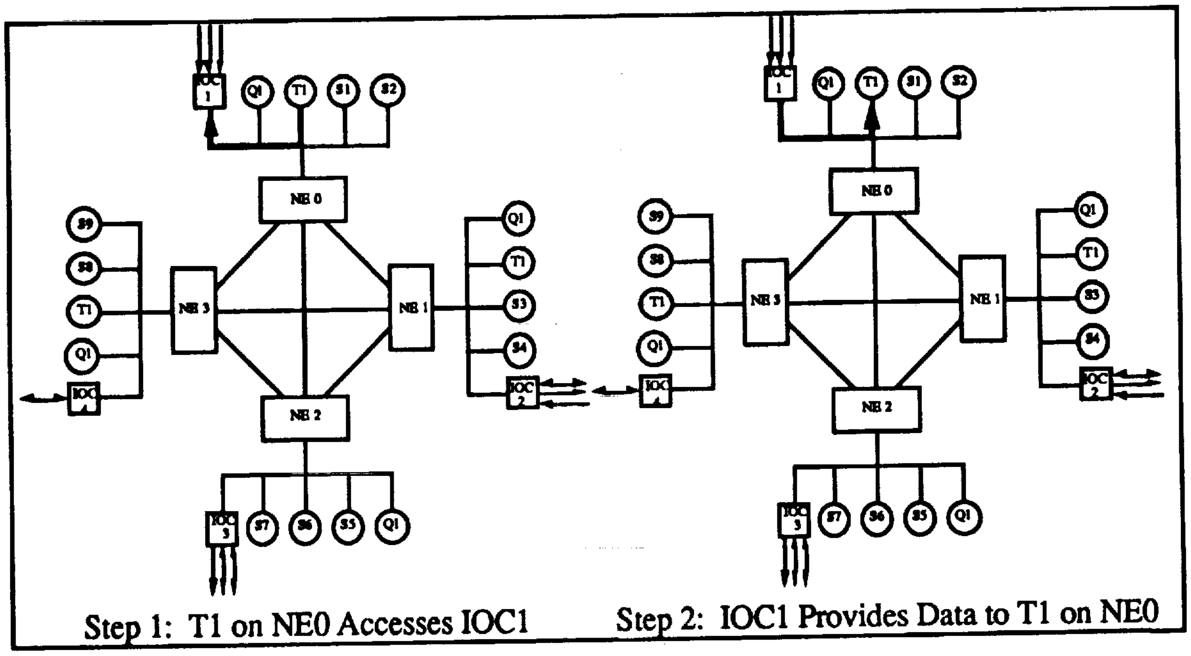

IOC 1
Q1
T1
S1
S2
NE 0
S9
S8
T1
Q1
IOC 4
NE 3
NE 1
Q1
T1
S3
S4
IOC 2
NE 2
IOC 3
S7
S6
S5
Q1
Step 1: T1 on NE0 Accesses IOC1
Step 2: IOC1 Provides Data to T1 on NE0

Figure 3-25. Steps 1 and 2 - T1 on NE0 Accesses IOC1 to Read Input Data

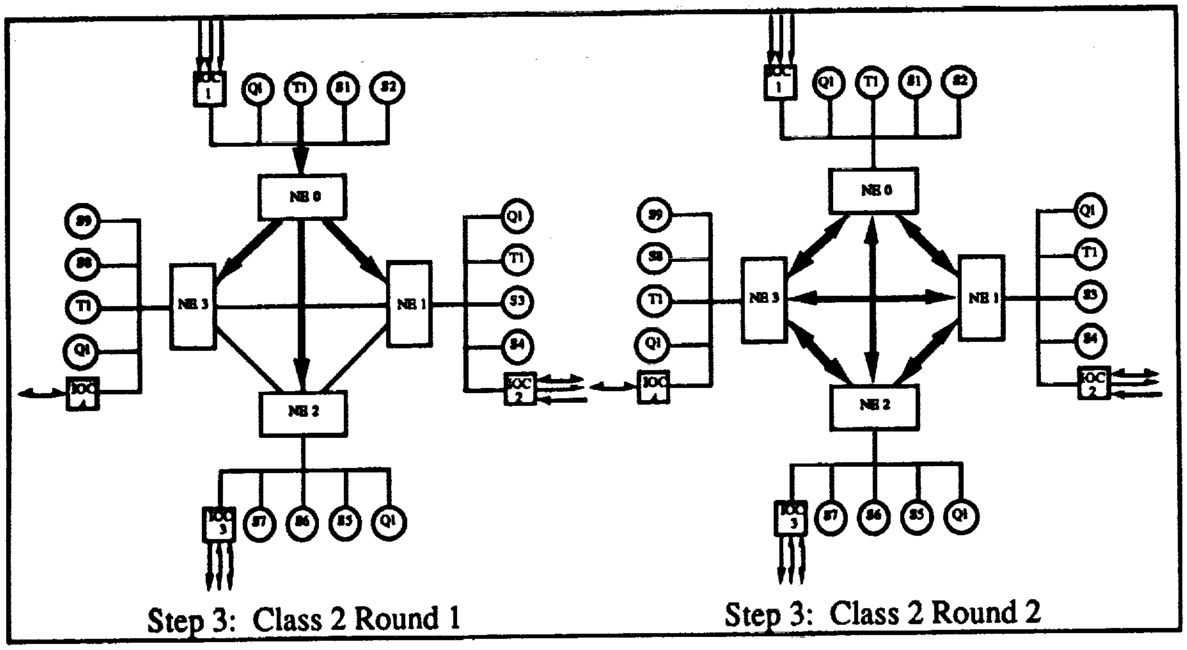


Figure 3-26. Step 3 - T1 Performs Class 2 Exchange, Rounds 1 and 2

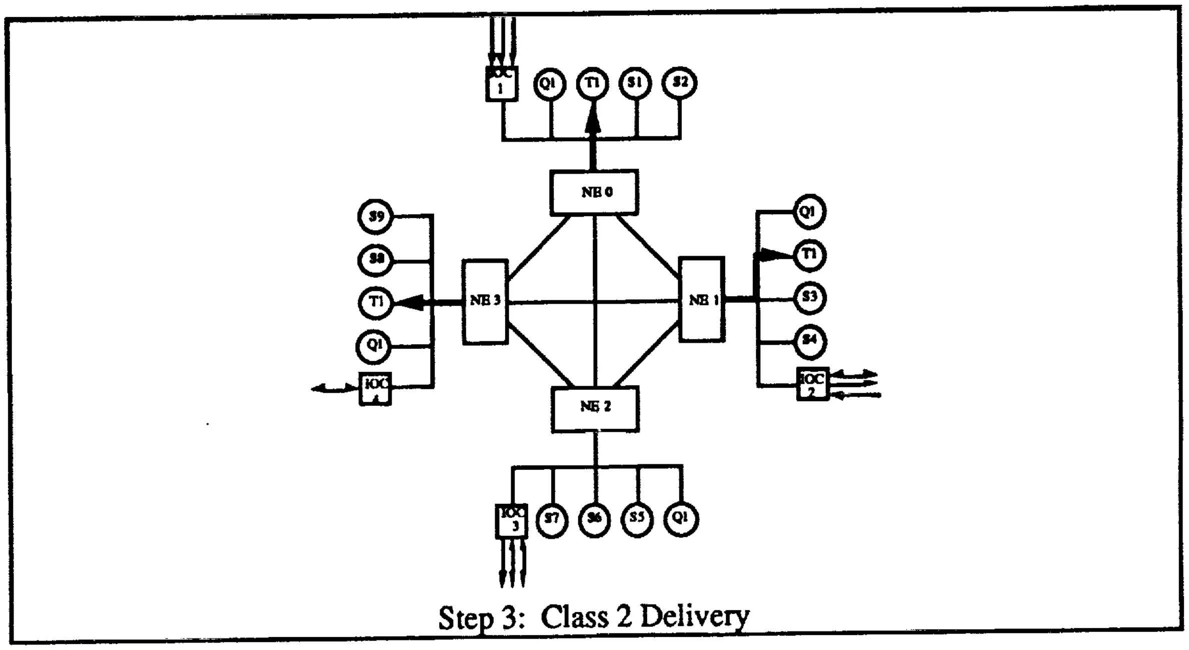


Figure 3-27. Step 3 - T1 Performs Class 2 Exchange, Delivery of Input Data to T1

3.4.8.1.6. Case 6: Dumb IOC, Simultaneous Input Reads by Triplex, Triplex Destination

In the previous example, only one channel of T1 (the one resident on NE0) accesses an input device, followed by a Class 2 Exchange from that channel to distribute the input data to the remaining members of the triplex T1. In some cases, it may be efficient for all members of T1 to concurrently access input devices resident in their local FCRs and perform a sequence of Class 2 Exchanges to distribute the data. These data may or may not necessarily represent redundant inputs. The idea here is that they are all accessed at approximately the same time, thus achieving a degree of parallelized input. If the IOCs do in fact represent redundant input devices, it is likely that even if they were nonfaulty they would not agree on a bit-for-bit basis. Therefore even in this case the following procedure must be executed, followed perhaps by a sensor redundancy management algorithm (average, mid-value select, etc.) on the three input data sets.

Procedure

Step 1: All members of triplex source T1 access FCR-local I/O devices to request inputs. Thus, T1 on NE0 accesses IOC1, T1 on NE1 accesses IOC2, and T1 on NE3 accesses IOC4.

Step 2: IOC1 provides the data to T1 in NE0 over the FCR backplane bus in FCR0, IOC2 provides data to T1 on NE1 over the FCR backplane bus in FCR1, and IOC4 provides the data to T1 in NE3 over the FCR backplane bus in FCR3.

Step 3: T1 on NE0 performs Class 2 Exchange to deliver its input data to its destination, say T1.

Step 4: T1 on NE1 performs Class 2 Exchange to deliver its input data to its destination, say T1.

Step 5: T1 on NE3 performs Class 2 Exchange to deliver its input data to its destination, say T1.

T1 may now use the three data sets. Note that Steps 3, 4, and 5 cannot be performed concurrently.

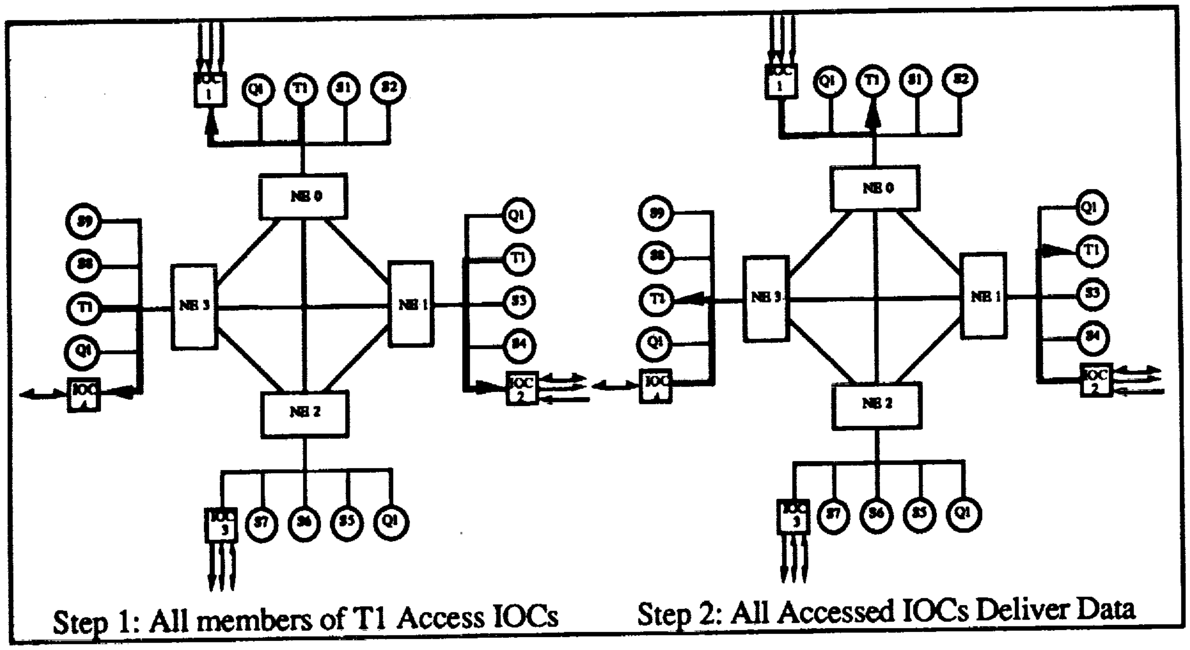

Step 1: All members of T1 Access IOCs
Step 2: All Accessed IOCs Deliver Data
NE 0
NE 1
NE 2
NE 3
IOC 1
IOC 2
IOC 3
IOC 4
Q1
T1
S1
S2
S3
S4
S5
S6
S7
S8
S9

Figure 3-28. Steps 1 and 2 - Members of T1 on NE0, 1, and 3 Simultaneously Access IOC1, IOC2, and IOC4 to Read Input Data

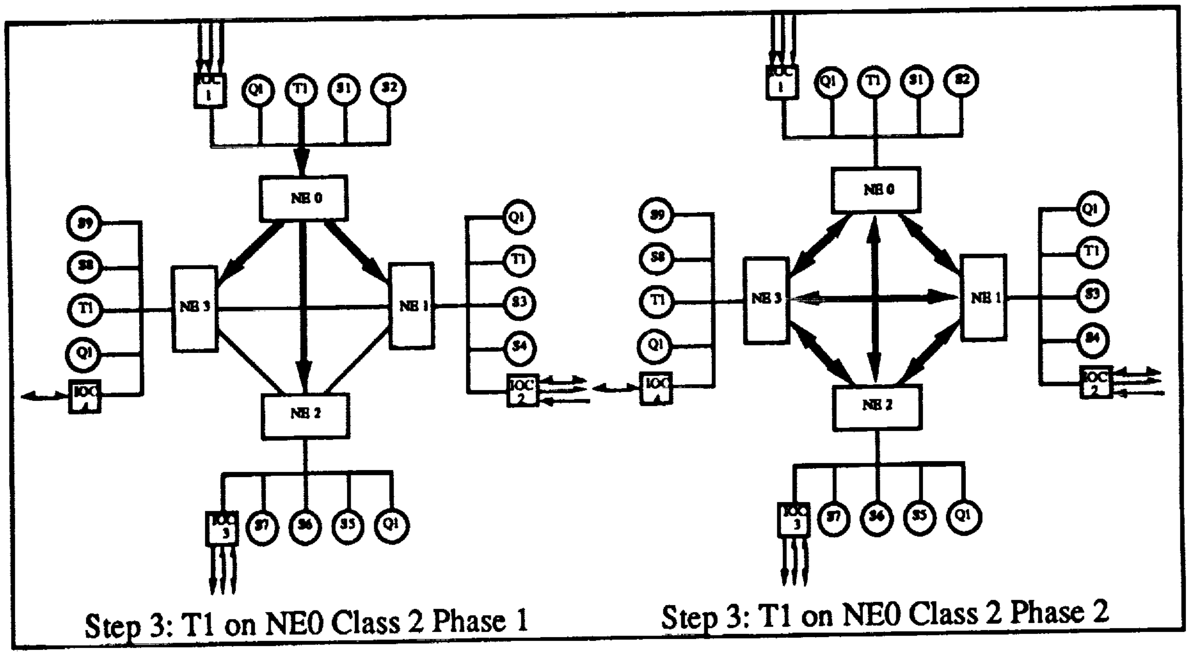

Step 3: T1 on NE0 Class 2 Phase 1
Step 3: T1 on NE0 Class 2 Phase 2
NE 0
NE 1
NE 2
NE 3
IOC 1
IOC 2
IOC 3
IOC 4
Q1
T1
S1
S2
S3
S4
S5
S6
S7
S8
S9

Figure 3-29. Step 3 - T1 on NE0 Performs Class 2 Exchange Phases 1 and 2

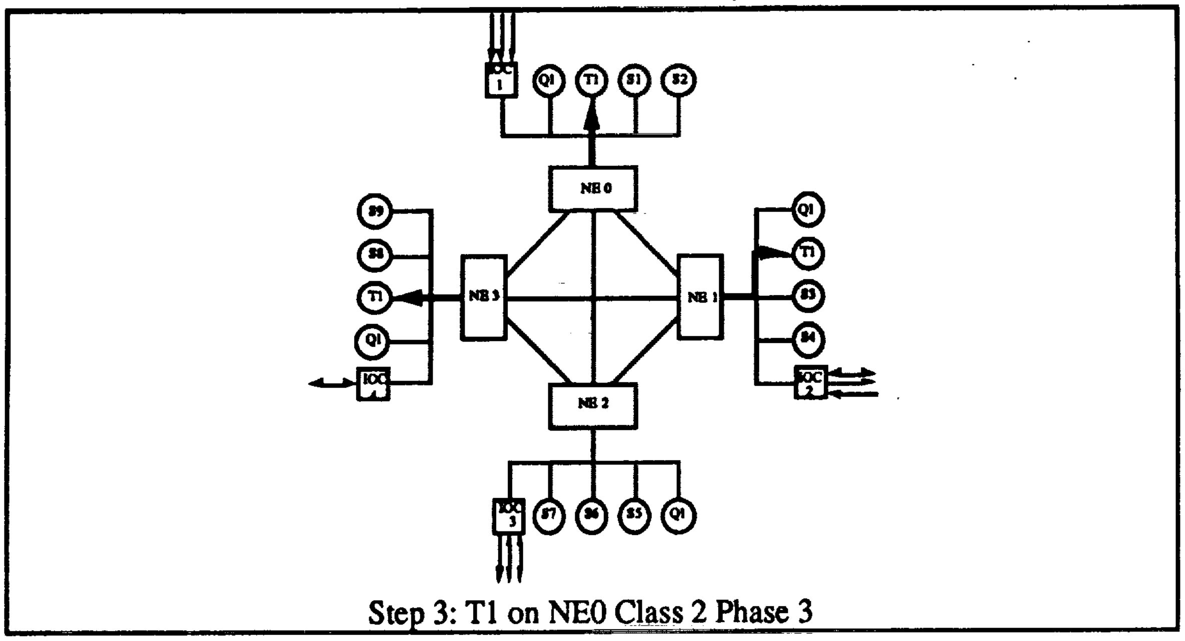


Figure 3-30. Step 3 - T1 on NE0 Performs Class 2 Exchange, Delivery of Input Data to T1

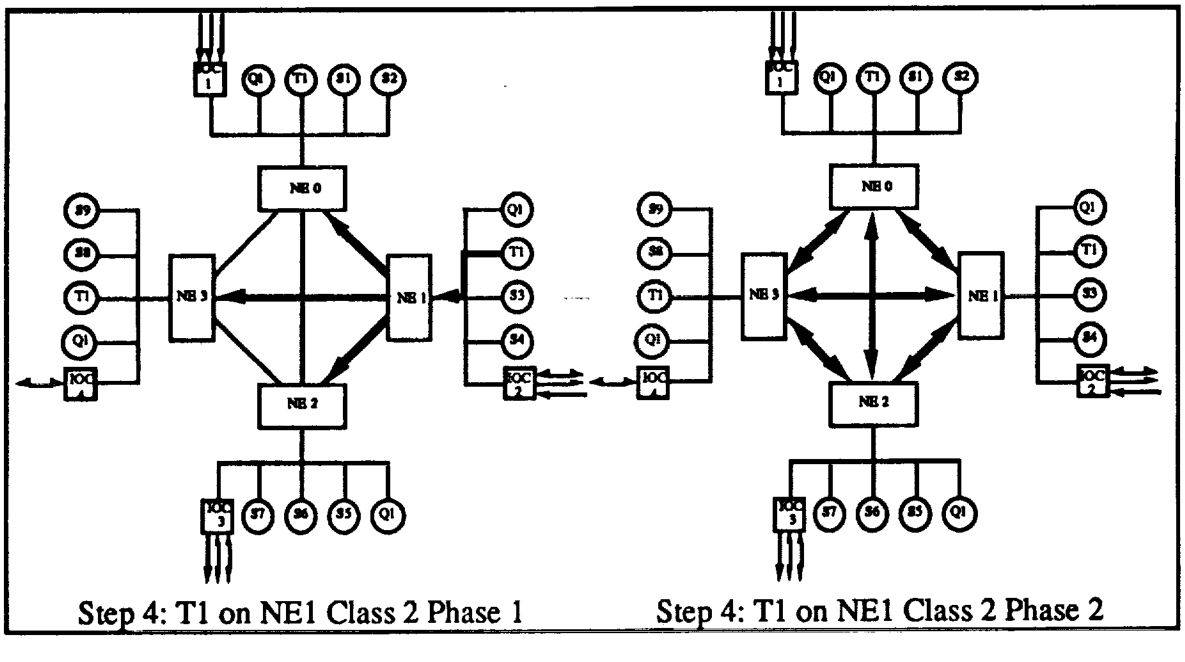


Figure 3-31. Step 4 - T1 on NE1 Performs Class 2 Exchange Phases 1 and 2

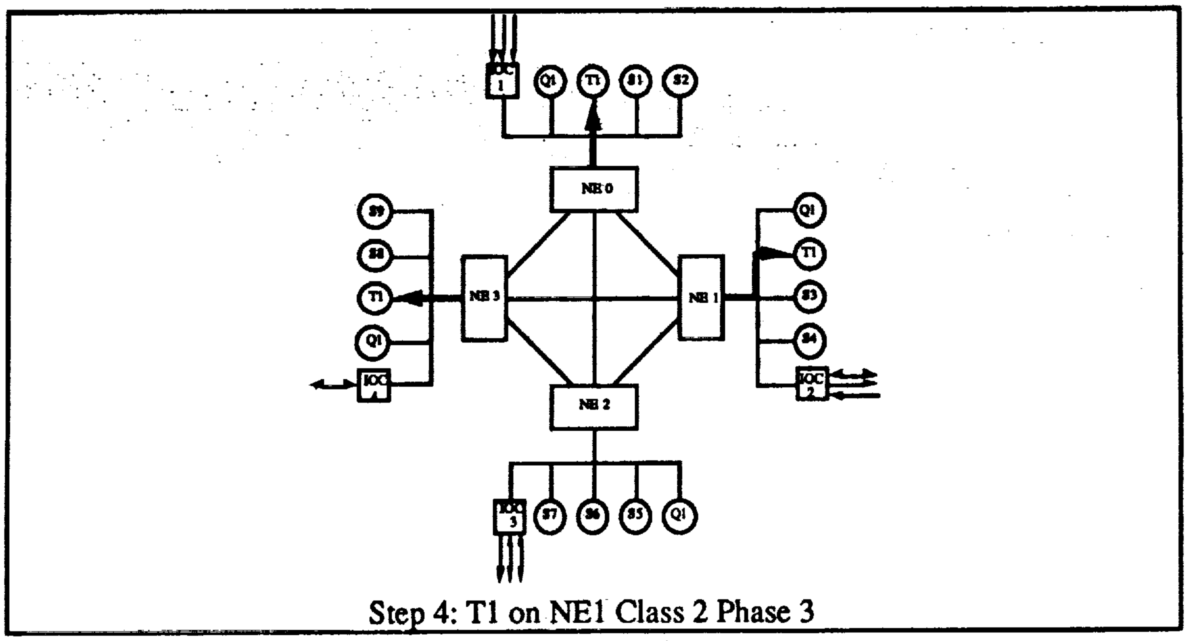


Figure 3-32. Step 4 - T1 on NE1 Performs Class 2 Exchange, Delivery of Input Data to T1

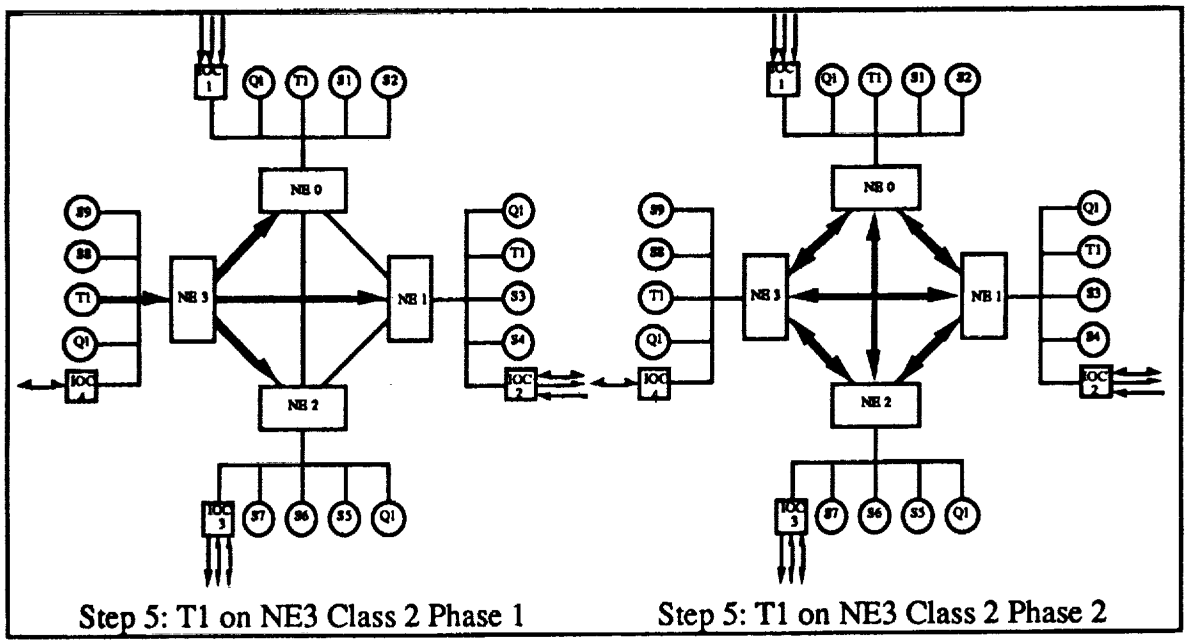


Figure 3-33. Step 5 - T1 on NE3 Performs Class 2 Exchange Phases 1 and 2

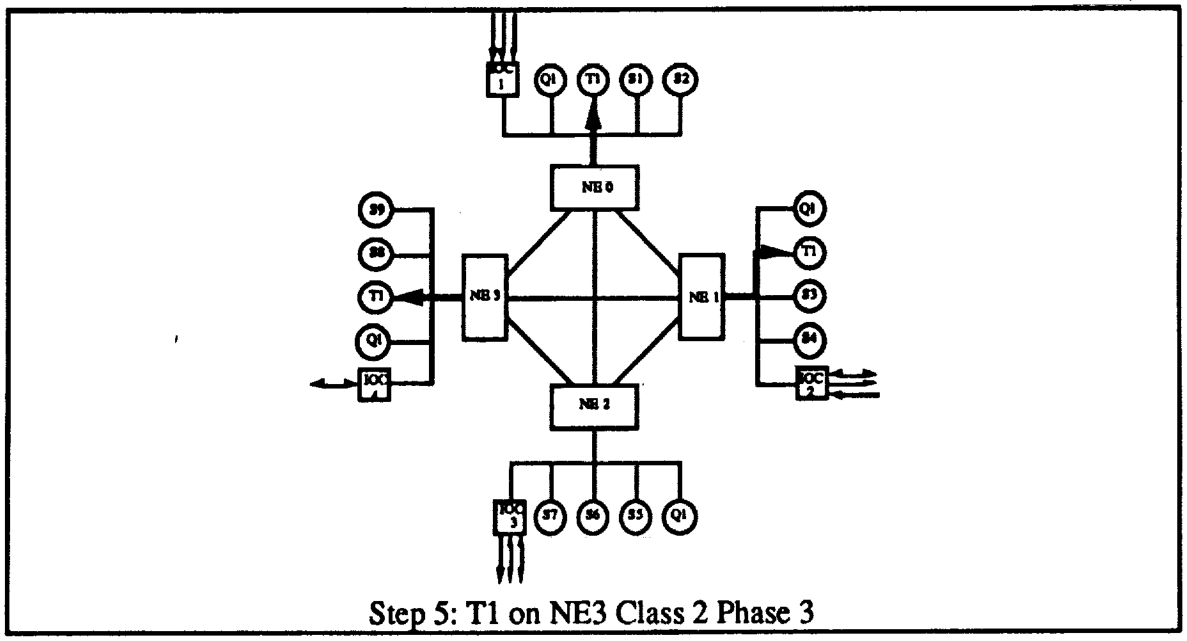


Figure 3-34. Step 5 - T1 on NE3 Performs Class 2 Exchange, Delivery of Input Data to T1

<u>3.4.8.2. Output Procedures</u>

A VG in the AFTA performs output by writing data or commands to an IOC either over the FCR backplane bus or over a local bus which is private to one or more PEs in a particular FCR.

3.4.8.2.1. Case 1: Simplex Source VG, Simplex IOC in same FCR as source VG

In this case, a simplex VG has an output datum it wishes to deliver to an IOC which resides in the same FCR as itself. No inter-FCR exchanges are necessary to achieve data consistency, so the simplex VG (say S9 in Figure 3-35) simply writes the datum to the corresponding IOC (say IOC4 in Figure 3-35). This is a non-fault tolerant operation and would not be used for critical outputs.

<u>Procedure</u>

Step 1: Simplex VG writes output data to simplex IOC over FCR backplane bus in FCR3.

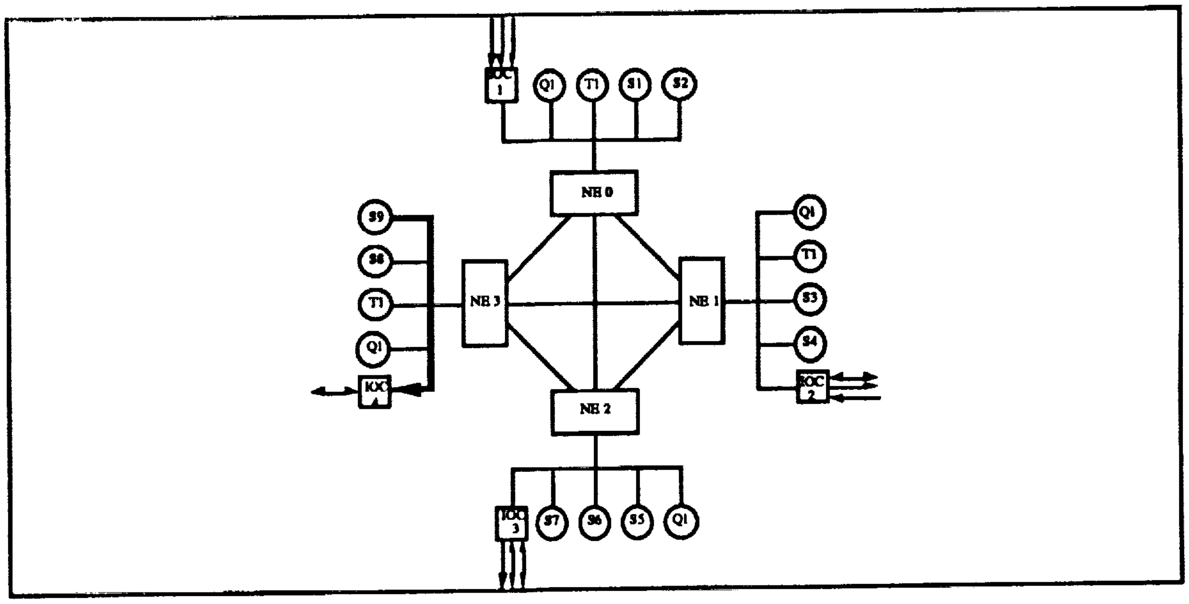


Figure 3-35. Simplex VG S9 Writes Output Data to Simplex IOC4

3.4.8.2.2. Case 2: Redundant Source VG, Simplex IOC in same FCR as one of VG Members, Unvoted Output Data

In this case, a triplex VG, say T1, must write output data to an IOC which is coresident in an FCR with a member of T1. In cases of noncritical outputs where the output data to be delivered to the IOC need not be voted to mask any computational faults the member of the VG closest to the IOC (say T1 on NE3) may write the data directly to the IOC (say IOC4 on NE3).

<u>Procedure</u>

Step 1: T1 on NE3 writes output data to IOC4 on NE3 over FCR backplane bus in FCR3.

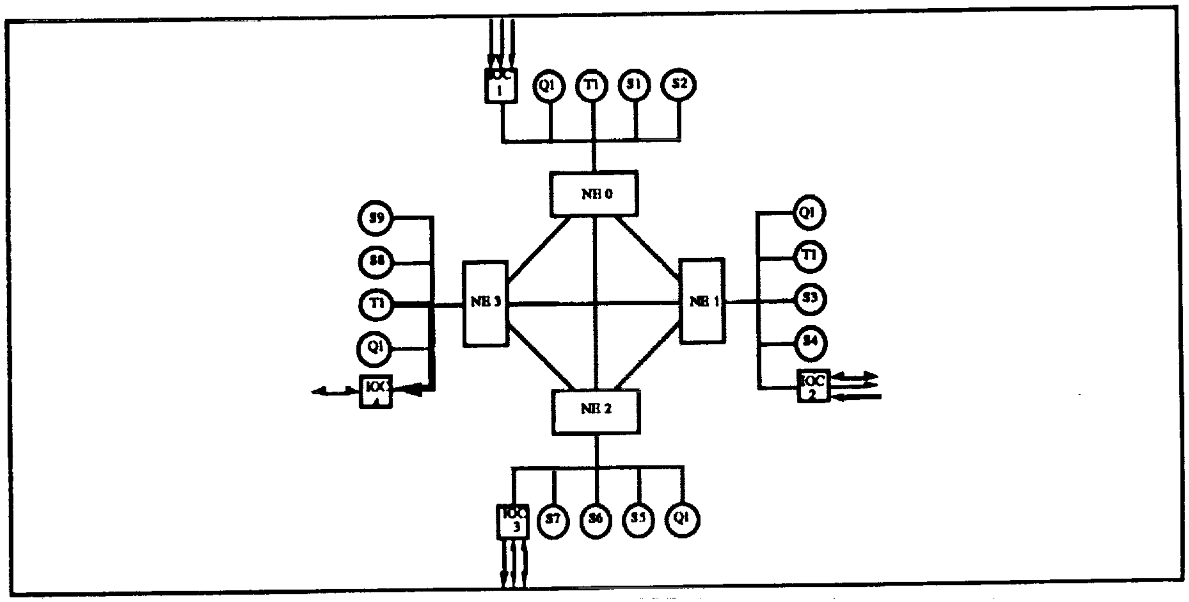


Figure 3-36. Triplex VG T1 Writes Unvoted Output Data to IOC4

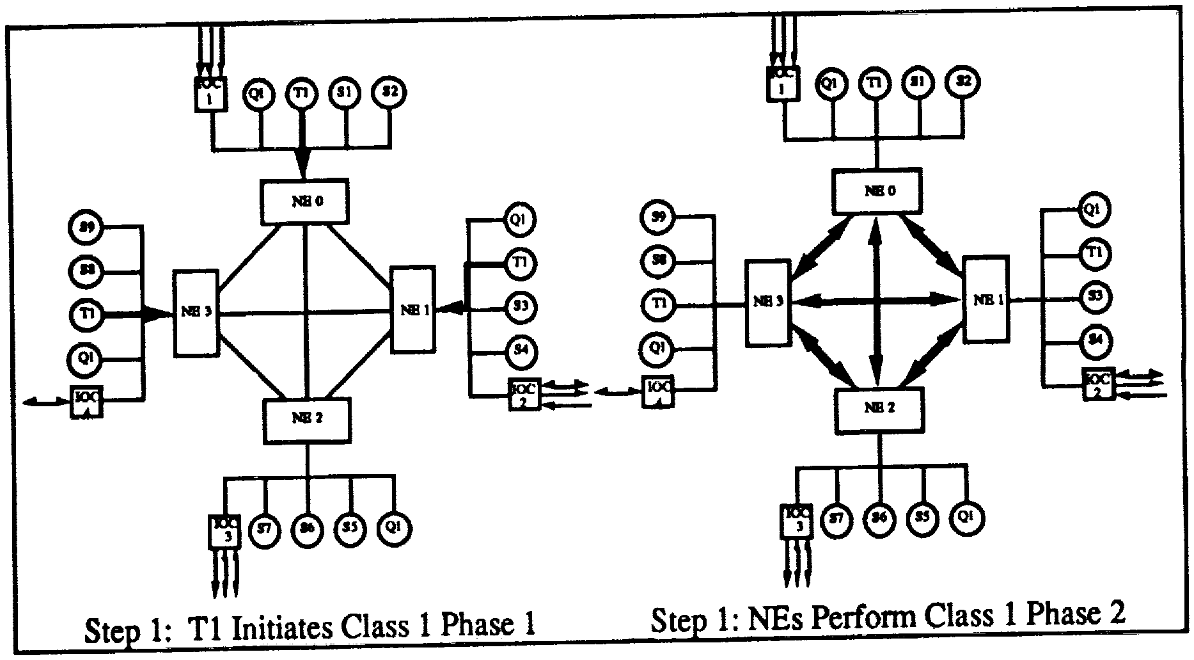
3.4.8.2.3. *Case 3: Redundant Source VG, Simplex IOC in same FCR as one of VG Members, Voted Output Data*

In the event that it is necessary to mask any faults occurring during the computation of the output to be delivered to an IOC, the redundant source VG must perform one or more Class 1 Exchanges prior to delivery of the output data to the IOC. Note that the actual process of delivering the output data to the IOC by its coresident VG member is not fault tolerant since either the IOC, the coresident VG member, the NE, or the bus interconnecting them all may be faulty.

Procedure

Step 1: Redundant source VG (T1 in Figure 3-37) performs Class 1 Exchange of data to be delivered to IOC (IOC4 in Figure 3-37).

Step 2: Member of redundant source VG coresident in same FCR as destination IOC (T1 on NE3 in Figure 3-38) writes voted output data to the IOC (IOC4 in Figure 3-38).



IOC 1
Q1
T1
S1
S2
NE 0
NE 1
NE 2
NE 3
S9
S8
T1
Q1
IOC 4
Q1
T1
S3
S4
IOC 2
IOC 3
S7
S6
S5
Q1
Step 1: T1 Initiates Class 1 Phase 1
Step 1: NEs Perform Class 1 Phase 2

Figure 3-37. Step 1 - Triplex VG T1 Votes Output Data, Phases 1 and 2

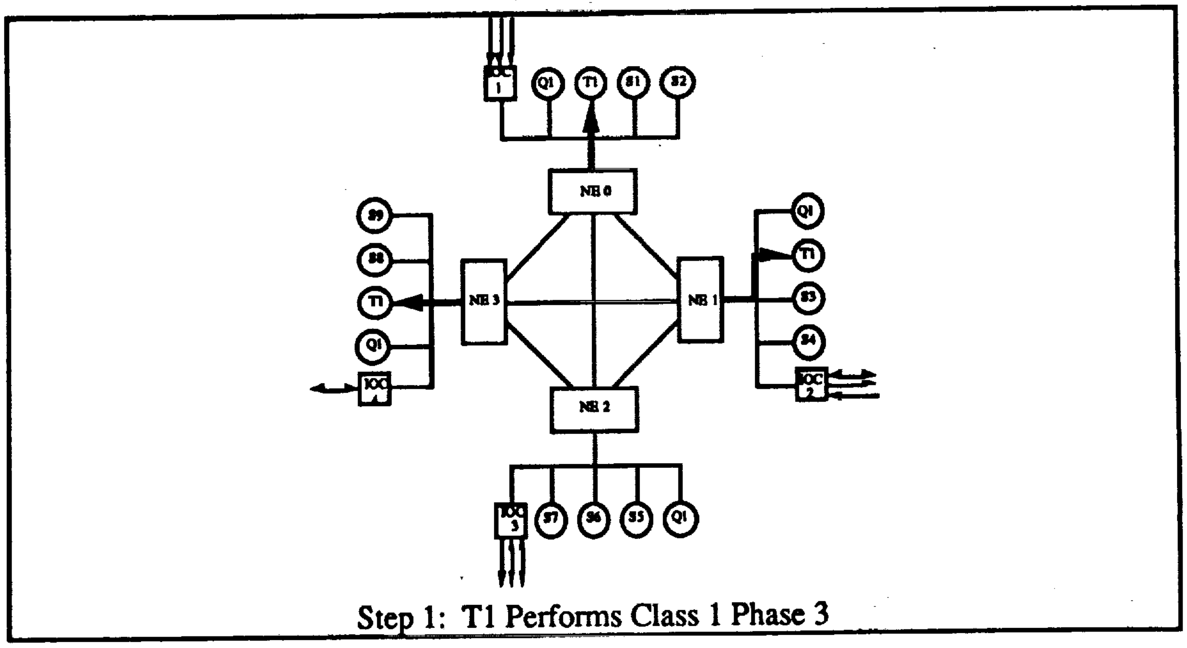


Figure 3-38. Step 1 - Triplex VG T1 Votes Output Data, Delivery of Voted Data to T1

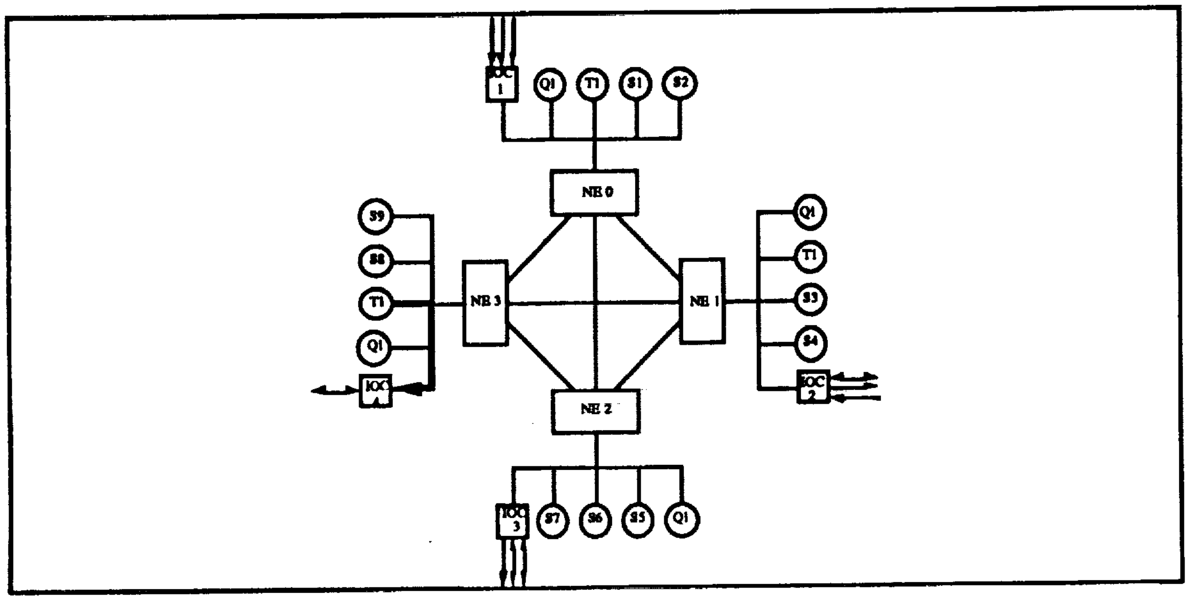


Figure 3-39. Step 2 - Triplex VG T1 on NE3 Writes Voted Output Data to IOC4

3.4.8.2.4. Case 4: Redundant Source VG, Redundant IOC in same FCR as one of VG Members, Unvoted Output Data

The previous cases dealt with outputs in which only one dumb IOC was driven by a coresident member of a VG. Since in all these cases the IOC and the coresident VG member were nonredundant, the corresponding procedure was not fault tolerant. To achieve fault tolerant outputs, redundant IOCs are necessary. In the case of redundant IOCs, it may be necessary that a redundant source VG provide outputs to multiple IOCs at very close to the same time. We illustrate with T1 providing near-simultaneous outputs to IOCs 1, 2, and 4. In this case, no validation of the computational output by voting the output data is performed by T1.

Procedure

Step 1: Members of redundant source VG T1 on NEs 0, 1, and 3 concurrently write output data directly to IOCs 1, 2, and 4, respectively, over the FCR backplane bus. Note that the data to be provided to the different IOCs need not be identical; such would be the case where for bandwidth reasons disparate outputs were to be asserted concurrently.

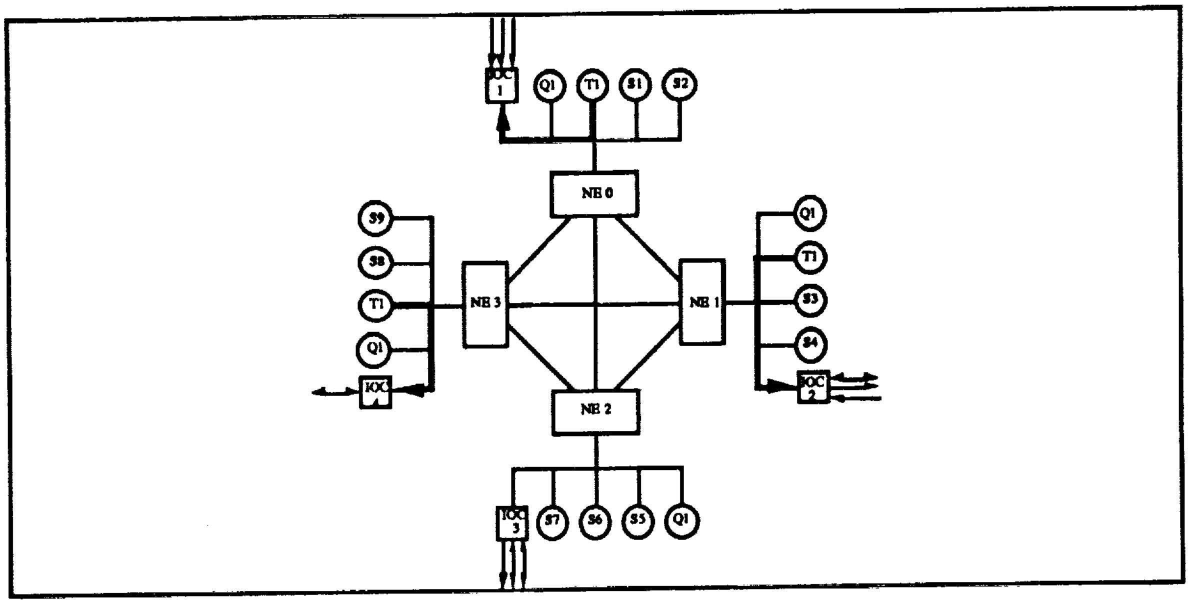


Figure 3-40. Redundant Source VG Simultaneously Writes Unvoted Output Data to Multiple IOCs

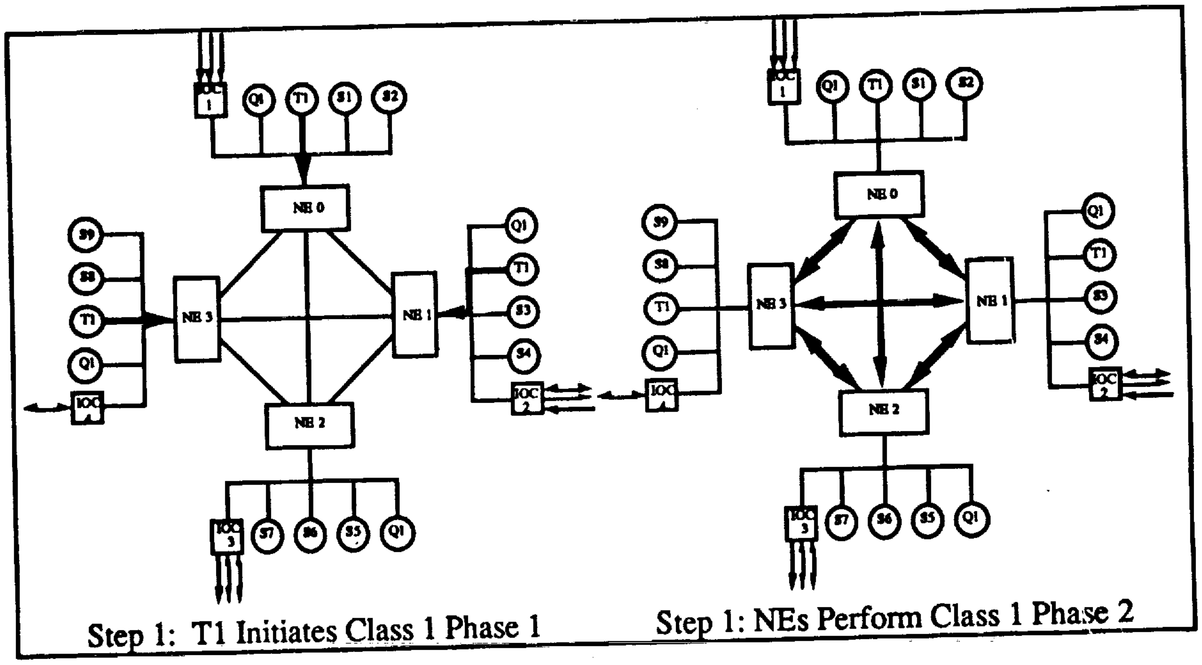
3.4.8.2.5. *Case 5: Redundant Source VG, Redundant IOC in same FCR as one of VG Members, Voted Output Data*

In cases where it is necessary to first mask computational faults prior to asserting the outputs, the redundant source VG must perform a Class 1 Exchange of the data to be output prior to writing the output data to the multiple IOCs. Again, note that the (now voted) data to be provided to the different IOCs need not be identical; the source VG would however have to separately vote the output data destined for each IOC.

<u>Procedure</u>

Step 1: Redundant source VG votes data destined for output IOCs.

Step 2: Redundant source VG members concurrently write output data to their respective IOCs over the FCR backplane bus.



IOC 1
Q1
T1
S1
S2
NE 0
S9
S8
T1
Q1
IOC 4
NE 3
NE 1
Q1
T1
S3
S4
IOC 2
NE 2
IOC 3
S7
S6
S5
Q1
Step 1: T1 Initiates Class 1 Phase 1
Step 1: NEs Perform Class 1 Phase 2

Figure 3-41. Step 1 - Triplex VG T1 Votes Output Data, Phases 1 and 2

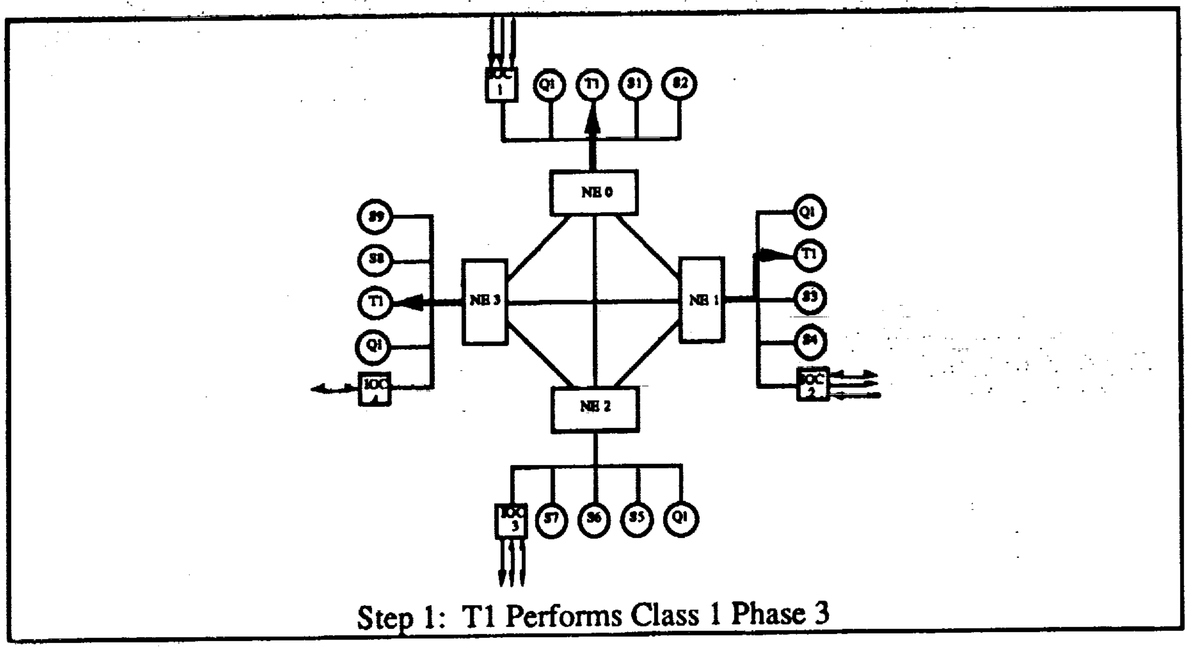


Figure 3-42. Step 1 - Triplex VG T1 Votes Output Data, Delivery of Voted Data to T1

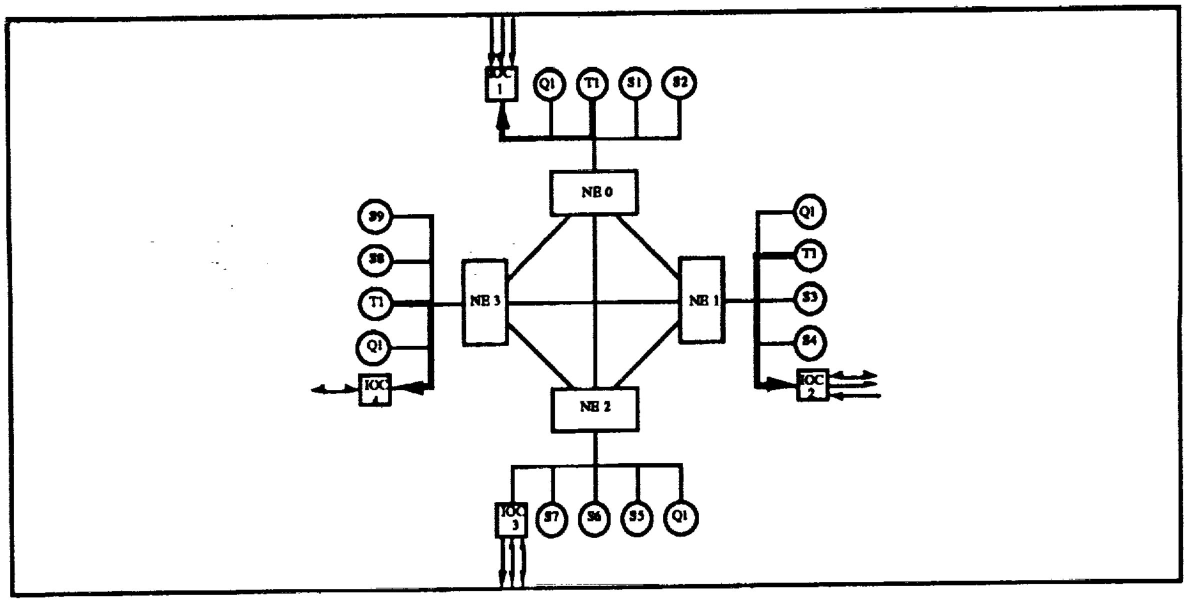


Figure 3-43. Redundant Source VG Simultaneously Writes Voted Output Data to Multiple IOCs

3.5. Operating System Architecture

The foundation of the operating system for the AFTA consists of a vendor-supplied Ada Run-Time System and CSDL-supplied extensions based on recommendations made by the Ada Run-Time Environment Working Group (ARTEWG). Additional features are required to manage the plurality of AFTA resources in a manner appropriate to the mission requirements. Services are provided in the following areas:

- Task management, i.e., task identification, scheduling, dispatching, suspension and termination
- Inter-task and interprocessor communication
- Input/Output services
- Management of parallel and redundant resources according to specified mission reliability, availability, and performance objectives
- Fault detection, identification, transient discrimination, and recovery throughout the maintenance and operational modes of the system

Additional services include software exception handling and time services, i.e., initialization of date and time from an external source, dissemination of current time and date, manipulation of times, conversion of the internal time representation to character format.

3.5.1. Scheduling

AFTA is designed for hard real-time applications. A Rate Group scheduler has been selected as the primary scheduling paradigm. This section discusses the rationale for the selection of this paradigm and illustrates its use to achieve hard real-time response for periodic and aperiodic hard real-time tasks.

3.5.1.1. Definitions

For the subsequent discussion the following definitions are in effect.

Hard Real-Time System: A hard real-time system is characterized by the presence of hard deadlines where failure to meet a deadline must be considered a system fault [SAE91].

In the AFTA context we consider the execution of hard real-time tasks, including input and output functions. The real-time constraints include the latencies from the system's input sensors to its output devices or actuators.

The timing constraints of a task can be specified in terms of one or more of the following parameters (similar to [Che87]).

Task Arrival Time: The task arrival time is the time at which a task is invoked by the system. Task invocation may be due either to the occurrence of an external event, the occurrence of a specified time, or the sampling of an input from an external source.

Task Deadline: A task's deadline is the time by which it must complete execution.

Periodic Task: A periodic task is invoked exactly once per period P. The arrival time of an execution of the periodic task specifies the time at which the execution of the periodic task is invoked. The arrival times and deadlines of a periodic task execution with constant period P are

$$A(i+1) = D(i) \tag{3.1}$$

$$D(i+1) = A(i+1) + P \tag{3.2}$$

where A(i) and D(i) are the arrival time and deadline of the i-th execution of the periodic task, respectively.

Jitter: Jitter is defined to be the difference between the minimum and the maximum completion times of an event such as task completion or I/O action, measured with respect to a fixed point in time such as a task arrival. For real-time control applications it is generally a quantity to be minimized.

Aperiodic Task: An aperiodic task, when invoked, is expected to execute exactly once, and has an arbitrary arrival time. Aperiodic tasks may or may not have hard deadlines with respect to invocation time.

Worst Case Computation Time: The worst-case computation time is an upper bound on the execution time of a task.

Static Priority Scheduling: In static priority scheduling the relative priorities of tasks are fixed prior to execution.

Dynamic Priority Scheduling: In dynamic priority scheduling the relative priorities of tasks may be changed during execution of the task suite.

Mixed Priority Scheduling: A mixed priority scheduling scheme includes both static and dynamic priority scheduling.

Preemptive Scheduling: A task is preemptible if its execution can be interrupted by other tasks and resumed afterwards. A preemptive scheduler supports such preemption.

Nonpreemptive Scheduling: A task is nonpreemptible if it must run to completion or self-suspension once it begins execution. A nonpreemptive scheduler does not support preemption.

3.5.1.2. Requirements for Hard Real-Time Schedulers

Hard real-time schedulers must ensure that task executions, inter-task interactions, and interactions between the tasks and the outside world are predictable and deterministic, with guaranteeable worst-case response time. The means for validating this guaranteed response time must be an integral part of the scheduling paradigm. The scheduling paradigm should exhibit formal tractability to facilitate its formal specification and verification to reduce the occurrence of scheduler design and implementation errors. The scheduler should enforce the notion of "separation of concerns" to permit the combinatorially explosive validation of a complex application to be accomplished via the more tractable option of validating its constituent parts and their interactions. Guaranteeing these properties is often in direct conflict with programming and maintenance ease. An engineering tradeoff must be performed keeping in mind the disastrous ramifications of failure to meet a hard real-time deadline and the high life-cycle cost of software maintenance.

3.5.1.3. Related Work

Relevant developments influencing the design rationale of the AFTA scheduler are outlined below.

3.5.1.3.1. Rate Monotonic Scheduling

The classic [Liu73] developed and analyzed a fixed priority preemptive scheduling policy in which tasks with higher request rates are statically assigned higher priorities, a policy known as *rate monotonic priority assignment*. Under the assumption that tasks are periodic, independent, and have no intertask data dependencies, [Liu73] showed that this scheduling policy is optimal in that "no other fixed priority assignment rule can schedule a task set which can not be scheduled by the rate-monotonic priority assignment." For a task set of m such independent periodic tasks with fixed priorities, a least upper bound on processor utilization was shown to be

$$U(m) = m\,(2^{1/m} - 1) \quad (3.3)$$

which, for large m, becomes

$$U(\infty) \approx \ln(2) \approx 0.693 \quad (3.4)$$

If this utilization is not exceeded the tasks will always meet their deadlines. This bound has been shown to be conservative [Leh87]. In practice, task periods which are harmonic or nearly harmonic can result in utilizations which are often higher than 90%.

3.5.1.3.2. MARS (MAintainable Real-time System)

The MARS (MAintainable Real-time System) [Kop89] is a distributed fault-tolerant computer system for the control of hard real-time applications. It uses time-triggered static preemptive scheduling and other concepts similar to the AFTA's RG scheduler. Tasks are statically scheduled based on timer interrupts and their interactions with the outside world, such as intertask communication and input/output, are defined only at those interrupts. This concept is called "temporal encapsulation" in [Kop89]. Temporal encapsulation is similar to the concept of "segmentation," the dividing of system resources into discrete and well-defined units, where the size of the unit is based on various criteria particular to the resource under consideration [Sta87]. In a hard real-time system time is clearly one of the most important resources to segment.

In MARS, task execution times are upper-bounded using the technique described in [Pus89] prior to (automated) schedule generation [Foh89]. Upper-bounds on task execution time are absolutely necessary to achieve a validatable ultra-reliable hard real-time system. Unfortunately, under the current state of the practice, verification of maximum task execution times is almost universally relegated to post hoc testing and validation. This practice can not provide statistically adequate guarantees that hard real-time deadlines will be met. In MARS, execution time bounding is facilitated since temporal encapsulation ensures that no delays due to communication and synchronization can occur during task execution. Prior to integration into a distributed system, each task is tested both in value and time domains. Because of temporal encapsulation, one is assured that a task's behavior when tested alone is identical to its behavior when integrated into the distributed system. The MARS scheduling paradigm was demonstrated for a distributed hard real-time application [Kop91] and it was found that

"Since the tasks have already been pretested in the domains of value and time, the system integration was very smooth.. ...The considerable test effort, which normally occurs during system integration, all but disappeared in this project. We assume that the temporal encapsulation of the MARS components...leading to a minimal coupling between the components, [is] responsible for this ease of implementation [Sch91]. Adding additional functions...would be realized and tested within a few hours."

3.5.1.3.3. IAPSA

Under the Integrated Airframe/Propulsion Control System Architecture (IAPSA) program ([Coh90a], [Coh90b])the Draper Advanced Information Processing System (AIPS) [Lal84] was evaluated as a candidate for integrated flight and propulsion control of a twin-engine advanced fighter aircraft. The Draper Fault Tolerant Processor (FTP) was used as the hardware platform and the AIPS Local System Services (LSS) [Bur89] were used to provide scheduling of the tasks. The AIPS LSS provides an extremely flexible task scheduling environment. Tasks can be scheduled according to priority, time, and event occurrence. Each task possesses a static priority and proceeds in a "run-until-blocked" mode until one of the following conditions occurs:

1. task completion;
2. self suspension;
3. preemption by hardware interrupt;
4. explicit scheduling of higher priority task(s); or
5. time slicing of an equal priority task.

A task can schedule itself or other tasks according to time (one-shot or periodic) or upon the occurrence of a software-defined event. The region of a task's periodicity can be bracketed by time or events, i.e., via start/stop times or events. Extensions to the Ada run time system were made to allow a task to schedule itself or another task cyclically, as the result of an event, at an absolute time, immediately, or to deschedule a task.

3.5.1.3.4. Reliable Computing Platform

NASA Langley Research Center is developing a reliable computing platform (RCP) [DiV90] to facilitate the development and demonstration of tools and techniques to support the design of a fault tolerant computing platform for the execution of flight-critical functions such as digital flight control. The RCP dispatches hard real-time control law application tasks and executes them on redundant processors. The entire RCP is described using a hierarchy of formal specifications - these levels include the uniprocessor model, the fault tolerant synchronous replicated model, the fault tolerant asynchronous replicated model, and the hardware/software implementation. Formal verifications are used to rigorously demonstrate a correct correspondence between any two adjacent levels of the description hierarchy.

The most important requirements of the control law applications for which the RCP is intended are stated to be:

1. the application comprises a fixed set of tasks;
2. the tasks have hard deadlines;
3. multi-rate cyclic scheduling is required;
4. upper bounds exist on task execution time; and
5. tasks must communicate using some form of interprocess communication.

In [DiV90] it is argued that

> "The design philosophy of the RCP is to design the system in a manner that minimizes the amount of experimental testing required and maximizes the ability to mathematically reason about correctness. The following design decisions have been made toward that end: ...the system is frame-synchronous [and the] scheduling is static, non-preemptive..."

The use of the static non-preemptive scheduler is further justified as follows:

> "...unfortunately the theoretical results can not guarantee that the hard deadlines will be met for any of the non-static or preemptive algorithms capable of scheduling the real-time control application tasks [McE88]. Consequently, all commercial aircraft control systems have been implemented using a static, non-preemptive schedule table."

3.5.1.3.5. NASA Space Transportation System General Purpose Computer

The Space Transportation System (STS) General Purpose Computer (GPC) hosts perhaps the most complex flight computer application ever developed. The two operating system scheduling approaches considered for the STS GPC ([Car84], somewhat superseded by [Han89]) were a synchronous concept and an asynchronous priority-driven concept. The synchronous approach was perceived to provide repeatability, predictability, and visibility into system operations, attributes which ease system verification and validation, but at the expense of adaptability for future growth. The asynchronous concept was viewed as readily accommodating growth, but more difficult to verify because it was not as predictable or repeatable as the synchronous scheme. The concept finally selected for the primary system software was a hybrid approach which used a synchronous foreground executive structure and a background priority-driven asynchronous dispatcher. For the guidance, navigation, and control functions, three rate groups are scheduled under the foreground executive with iteration frequencies of 25 Hz for support of the basic vehicle flight control, 6.25 Hz for intermediate-frequency functions, and 0.25 Hz for the display update function.

3.5.1.4. AFTA RG Scheduler Overview

The AFTA supports two different styles of scheduling. The first, known as *rate group scheduling*, is suitable for task suites in which each task has a well-defined iteration rate and can be validated to have an execution time which is guaranteed to not exceed its iteration frame (the inverse of its iteration rate). A modification of rate group scheduling discussed below also allows aperiodic hard real-time events to be processed. The second style of scheduling, known as aperiodic non-real-time scheduling, is available when the iteration rate of a particular non-real-time task is unknown or undefined. Validation of the temporal behavior of such tasks may be difficult. In AFTA, non-real-time aperiodic tasks are not allowed to perturb the critical timing behavior of rate group tasks.

The basic executive of the AFTA is the XDAda RTS enhanced by CSDL. One of these enhancements is a multi-rate group scheduler layered upon the XDAda run time executive. In such a paradigm tasks executing on each VG in the AFTA are characterized by an iteration rate. In the AFTA, these rates are nominally 100, 50, 25, and 12.5 Hz, corresponding to rate group identifiers R4, R3, R2, and R1, respectively*. A rate group frame duration is the inverse of the rate group iteration rate; thus the R4, R3, R2, and R1 frames are 10, 20, 40, and 80 ms in duration, respectively. All frame boundaries are determined by crystal oscillator-controlled interrupts, as described below. The frequencies and number of rate group frames are readily changed as the application dictates. Frames executing on different VGs in the AFTA need have no particular phase relationship with each other, although a desired phase relationship among certain frames may be enforced in some applications using the phasing method described below.

Within a particular rate group frame, tasks are scheduled using a nonpreemptive static schedule. When scheduled, a task executes to self-suspension. The exact time of execution of a particular task in the rate group frame will be in general unknown to the applica-

* The origin of this unfortunate nomenclature is lost in the mists of antiquity.

tion programmer. Instead, AFTA guarantees that all tasks within a rate group will be executed in the order specified by the application programmer sometime within the appropriate rate group frame. Figure 3-44 illustrates the basic idea of a single rate group.

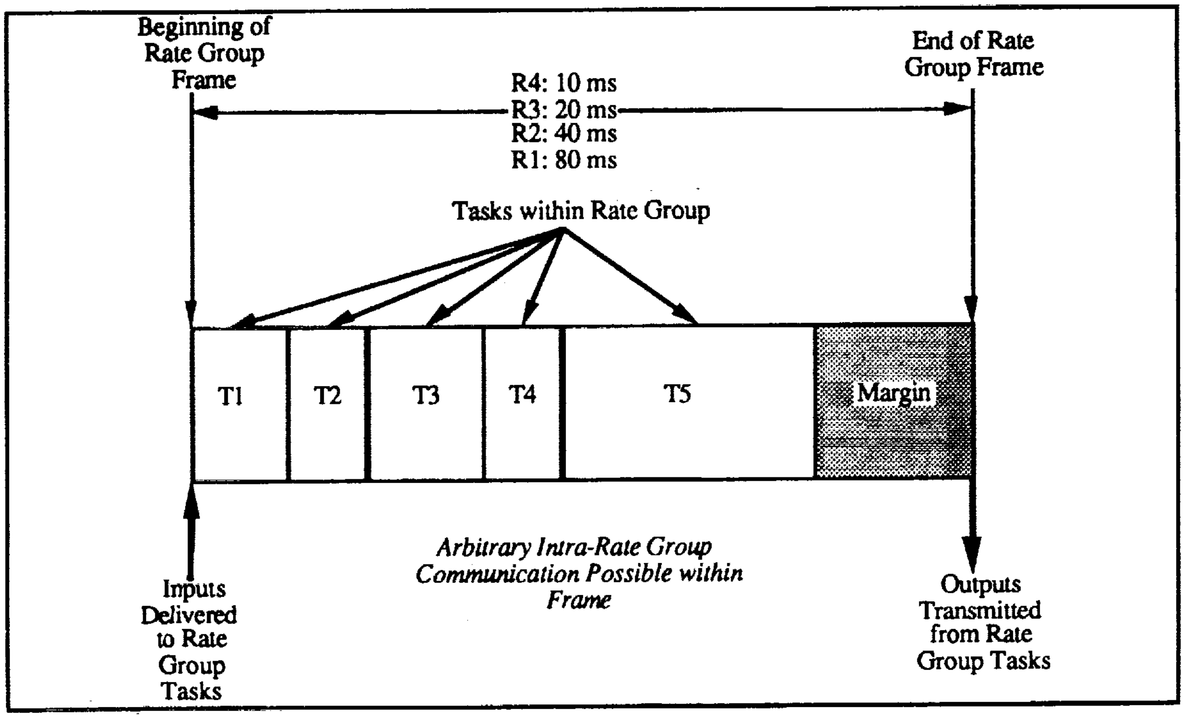

Beginning of Rate Group Frame
End of Rate Group Frame
R4: 10 ms
R3: 20 ms
R2: 40 ms
R1: 80 ms
Tasks within Rate Group
T1
T2
T3
T4
T5
Margin
Arbitrary Intra-Rate Group Communication Possible within Frame
Inputs Delivered to Rate Group Tasks
Outputs Transmitted from Rate Group Tasks

Figure 3-44. Rate Group Frame - Programming Model

To achieve multi-rate group execution on a VG, lower frequency rate group tasks are interrupted on a periodic basis to allow the higher-frequency rate groups to execute (Figure 3-45). The interruption process is transparent to the application programmer.

Task overruns are detected by the rate group dispatcher at the end of each RG frame. Since all tasks within a frame nominally execute to self-suspension, the rate group dispatcher can detect a frame overrun by checking the suspension status of tasks which should have completed an iteration in the preceding rate group frame. Note that since the task which caused the overrun may have completed in the frame yet caused a subsequently-scheduled task to overrun, this technique does not conclusively identify which task is responsible for the overrun. Identification of the culprit task is achieved by comparison of the actual measurement of each task's execution time with its predicted execution time (note that this information is already needed for construction of the task schedule). Several overrun handling options exist and must be selected on a task-specific basis. Examples include aborting or restarting the culprit task, or resuming the preempted task from its preemption point at the start of its next RG frame.

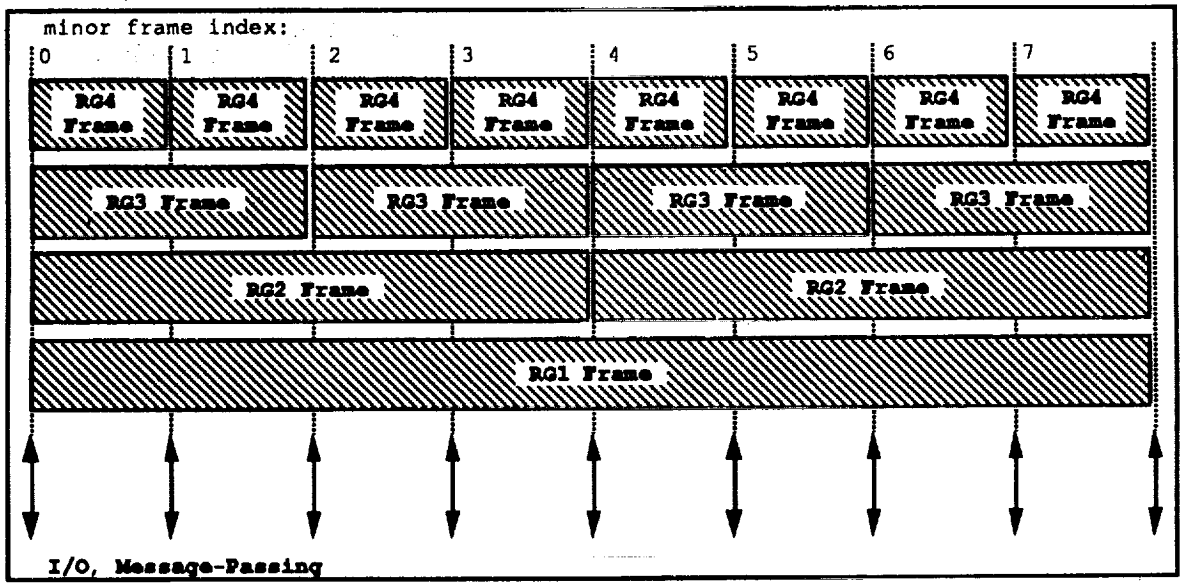


Figure 3-45. Architecture of RG Frames on a Single VG

Rate group scheduling may be viewed as a compromise between dynamic preemptive and static non-preemptive scheduling. Within a rate group, a static nonpreemptive schedule is followed. Higher frequency rate groups preempt lower frequency rate groups in a variant of rate monotonic scheduling [Liu73] modified for task suites having harmonic iteration frequencies. Because they interact only on frame boundaries, the set of rate groups may be viewed as a decoupled set of nonpreemptive tasks which may be formally treated independently.

3.5.1.4.1. Intertask Communication

All communication to tasks within a rate group, whether from input devices, the Network Elements, tasks executing in other rate groups on that VG, or messages emanating from other VGs in the AFTA, is delivered and made available to the rate group tasks at the beginning of their rate group frame, assuming it was sent in time to be received by the recipient VG before the frame boundary. All communication emanating from tasks within a rate group, whether sent to output devices, the Network Elements, tasks executing in other rate groups on that VG, or other tasks executing on other VGs, is queued within the rate group frame and transmitted at the end of that rate group frame. The single exception to this rule is made for non-preemptible R4 tasks, which can send and receive messages at any time. All messages not read by a RG task by the end of its frame can either be retained or deleted, with appropriate notification given to the recipient task.

3.5.1.4.2. Overview of Minor Frame

A simplified description of the sequence of events occurring within a minor frame of a single VG is depicted in Figure 3-46. The frame begins with a Frame Timer interrupt which is generated by a crystal oscillator resident on each member of the VG. Immediately after the Frame Timer interrupt, the VG synchronizes its members using a synchronizing act as described in Section 3.4.5, and sets up the Frame Timer interrupt for the next minor frame. This reduces the skew with which the members of the VG receive the next Frame Timer interrupt to the Network Element's post-synchronization skew plus the crystal oscillators' drift over the frame.

After the synchronizing act, the I/O Dispatcher performs all I/O activity as close as possible to the synchronizing act in order to minimize I/O jitter. For I/O performance reasons, it is possible for each member of a VG to perform different I/O transactions and thus not to be in synchrony after performing such operations. Therefore an I/O Completion interrupt is scheduled on all VG members at a known time after the Frame Timer interrupt in order to snap them back into synchronization. The Frame Timer - I/O Completion interrupt interval may vary for each frame based on the I/O transactions performed in that frame, and is determined by the most lengthy set of transactions the VG's members must perform. This interrupt is generated by a crystal oscillator on each VG member.

After the I/O Completion interrupt another VG synchronization is performed by the Task Dispatcher, and messages queued for transmission by rate group tasks which completed in the prior frame are transmitted to the Network Element. Messages are also read from the NE to the VG at this time, among others. The FDIR task (also known as Redundancy Management, or RM) is scheduled after message passing, followed by the I/O Source Congruency and Redundancy Manager and I/O Processing tasks. The I/O tasks are responsible for transmitting single-source input data from one member of the VG to the others, I/O Controller/Device error processing, and deriving and formatting a known good copy of redundant input data for delivery to the destination application task. The I/O Processing task is also responsible for transmitting predetermined input data from one VG to another.

After the I/O tasks execute, the application tasks are scheduled and execute according to the rate group scheduling paradigm until the next Frame Timer interrupt.

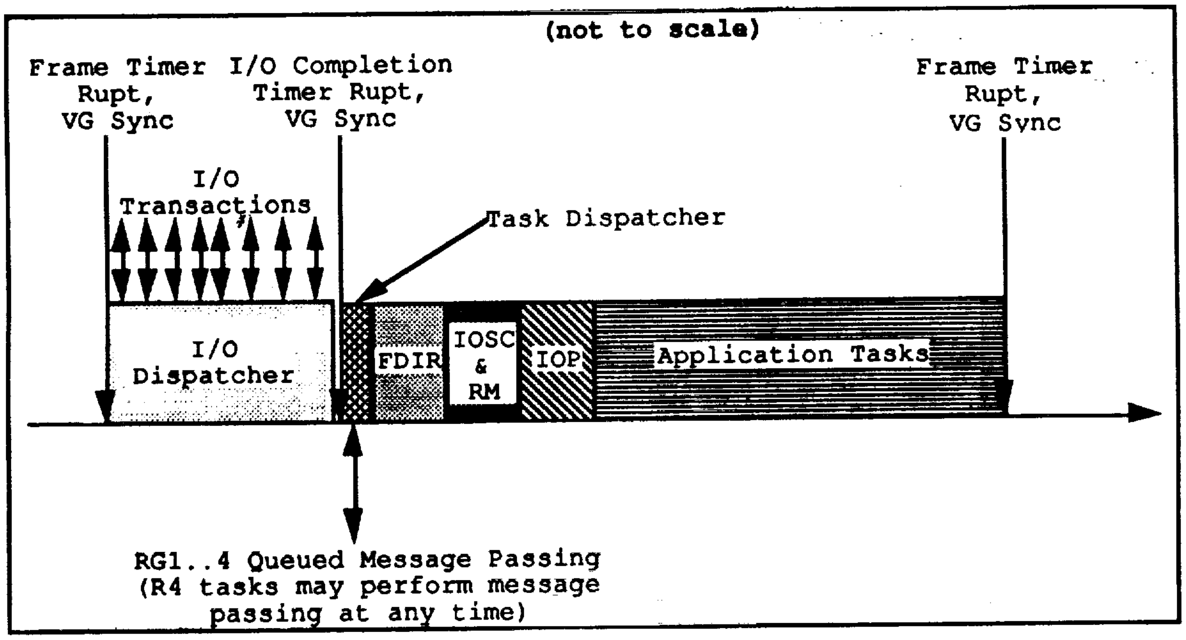

(not to scale)
Frame Timer Rupt, VG Sync
I/O Completion Timer Rupt, VG Sync
Frame Timer Rupt, VG Sync
I/O Transactions
Task Dispatcher
I/O Dispatcher
FDIR
IOSC & RM
IOP
Application Tasks
RG1..4 Queued Message Passing (R4 tasks may perform message passing at any time)

Figure 3-46. Overview of Minor Frame

3.5.1.4.3. Preemptive Rate Group Scheduling

The implementation of preemptive rate group scheduling is depicted in Figures 3-47 through 3-53. Figure 3-47 illustrates a schedule containing only the Dispatcher executing on a VG; Figure 3-48 illustrates a schedule containing the Dispatcher and the RM task; Figure 3-49 illustrates a schedule containing the Dispatcher, the FDIR task, and the R4 tasks executing on a VG. The remaining rate groups are presented similarly in Figures 3-50 through 3-52. Figure 3-53 contains a schedule showing the execution of all rate groups and background tasks.

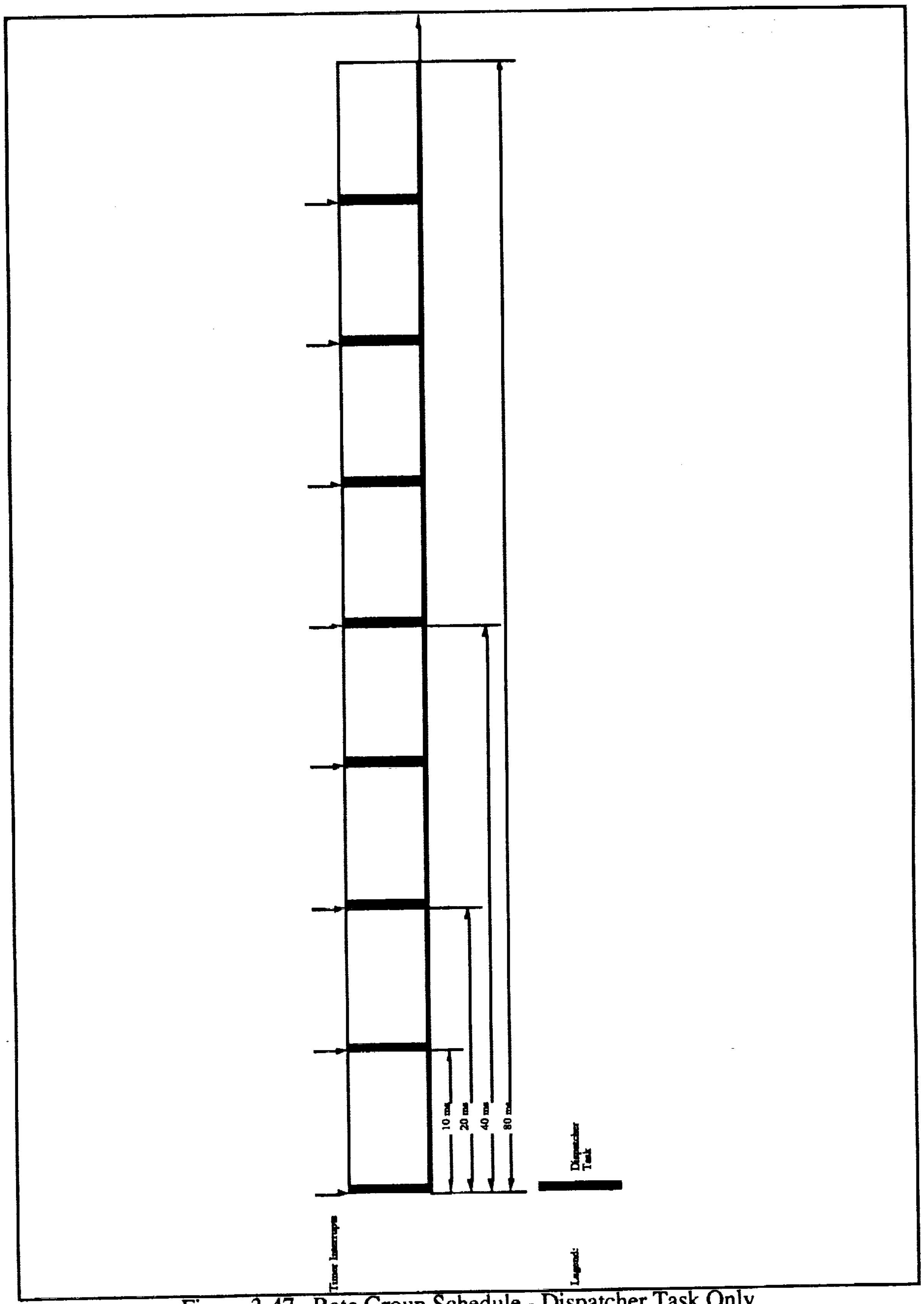


Figure 3-47. Rate Group Schedule - Dispatcher Task Only

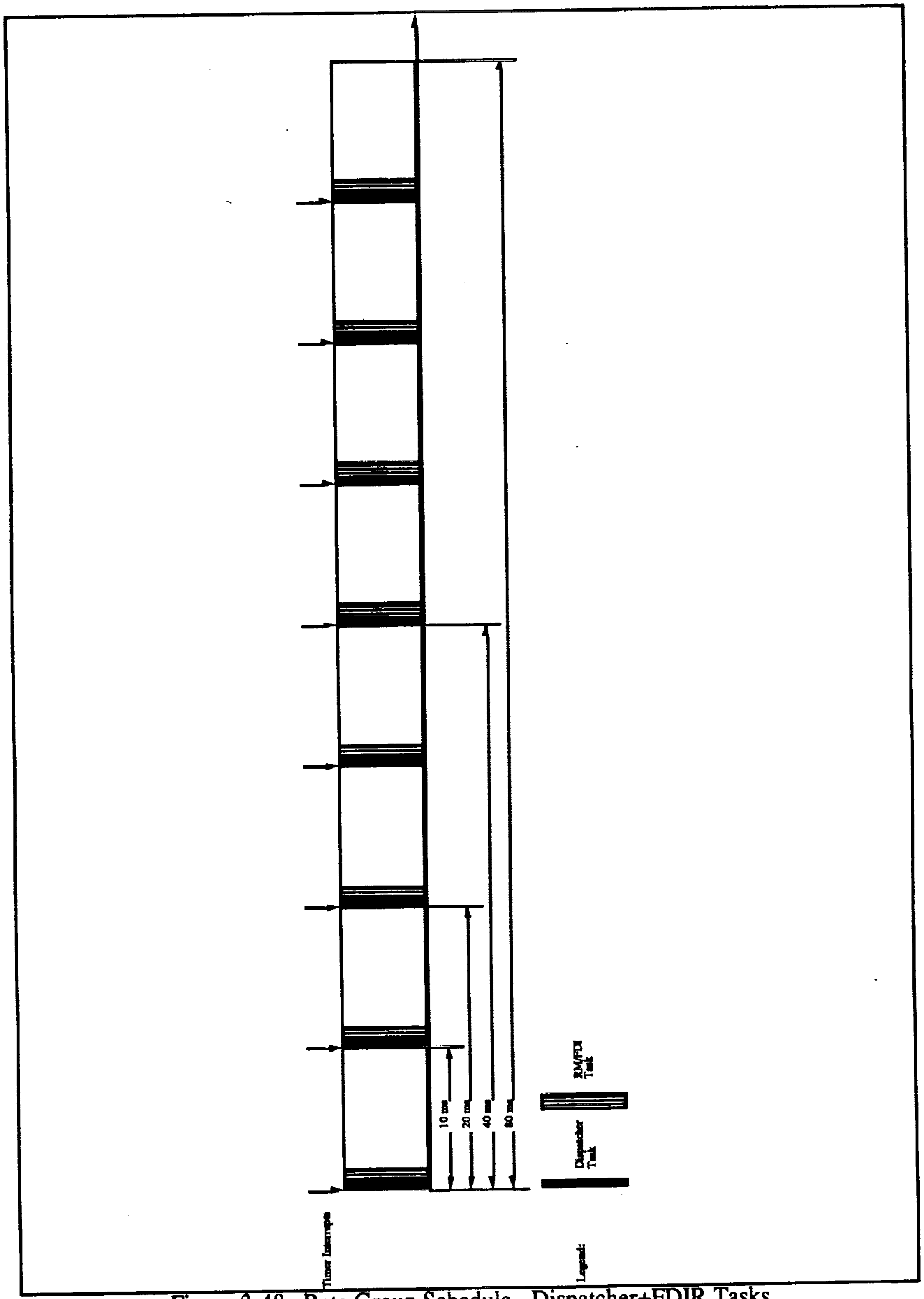


Figure 3-48. Rate Group Schedule - Dispatcher+FDIR Tasks

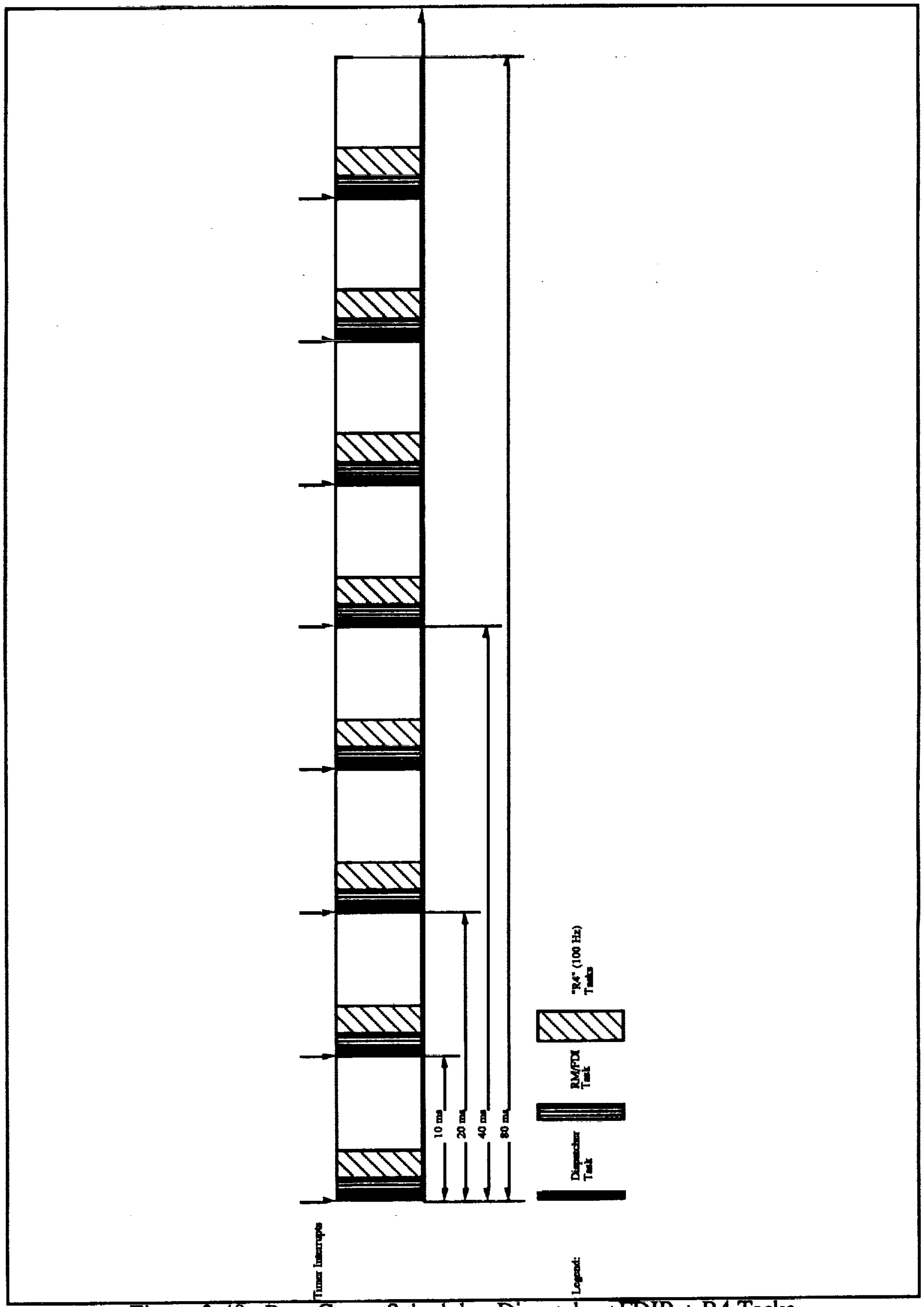


Figure 3-49. Rate Group Schedule - Dispatcher+FDIR + R4 Tasks

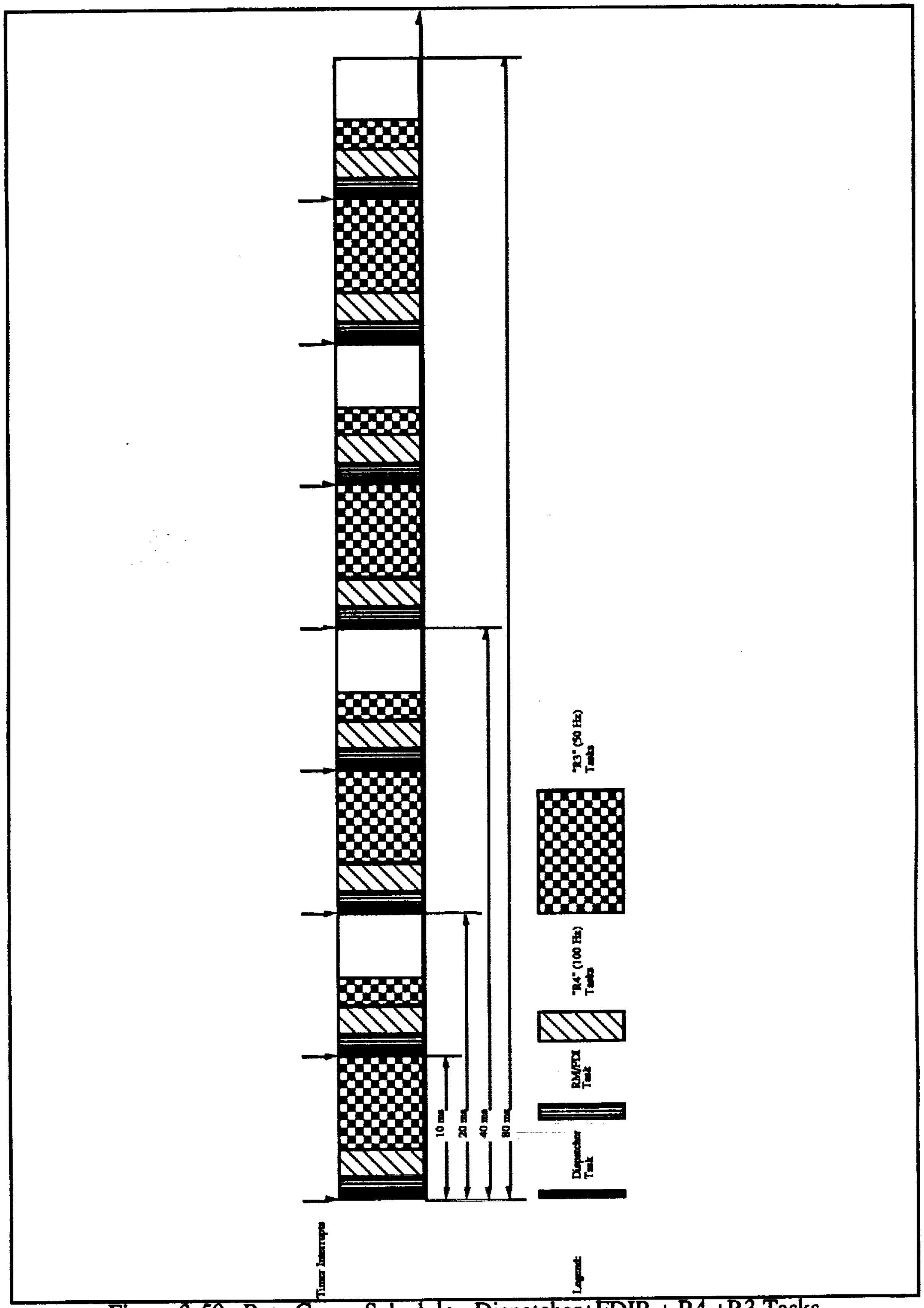


Figure 3-50. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 Tasks

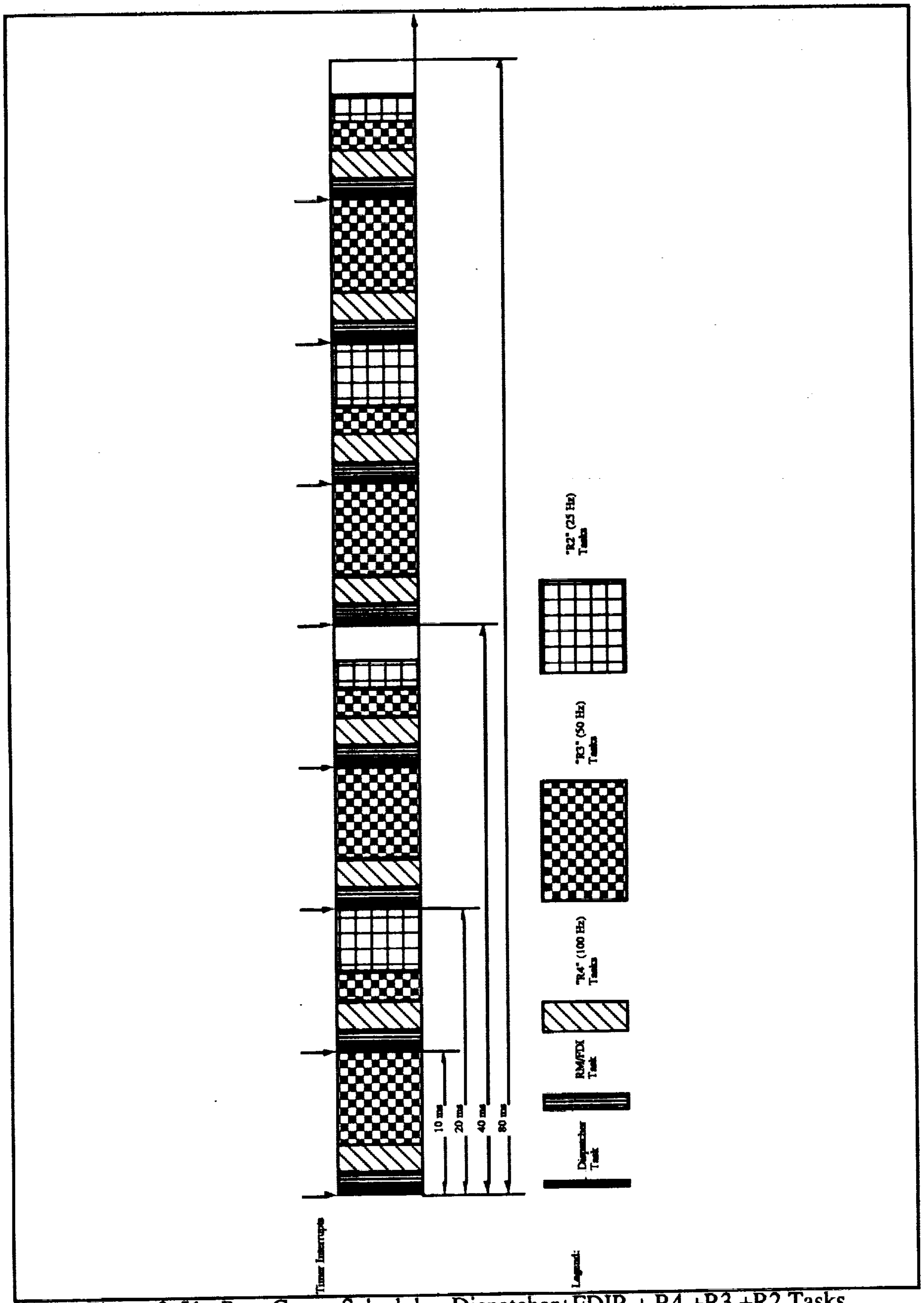


Figure 3-51. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 Tasks

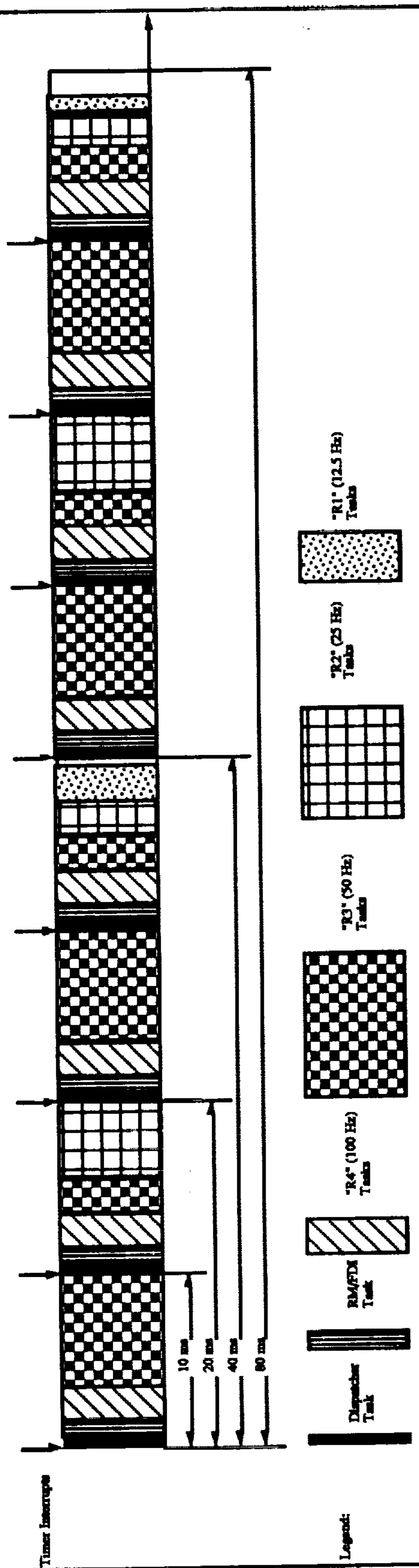


Figure 3-52. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 +R1 Tasks

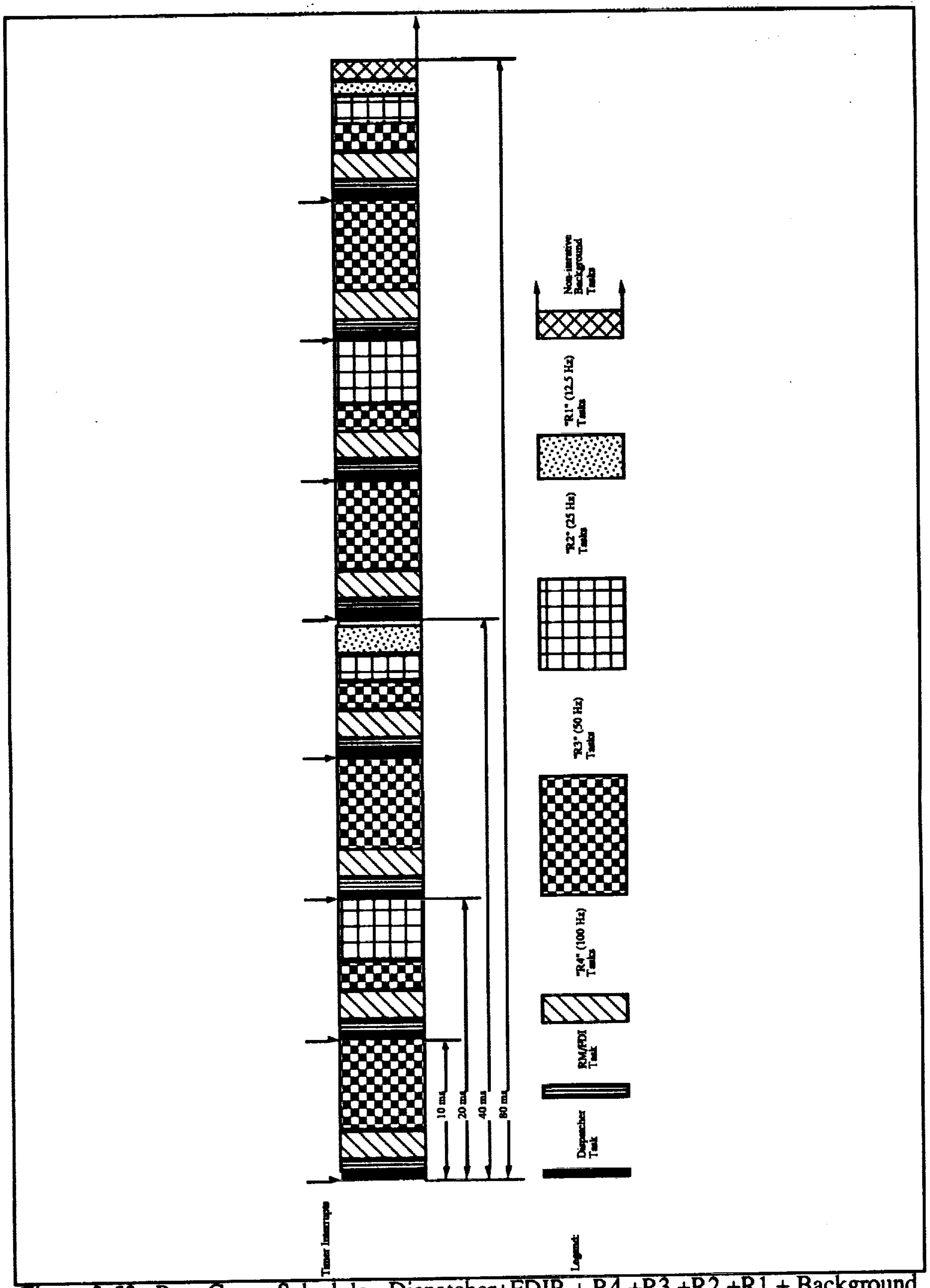


Figure 3-53. Rate Group Schedule - Dispatcher+FDIR + R4 +R3 +R2 +R1 + Background Tasks

3.5.1.4.4. *Aperiodic Hard Real-Time Task Scheduling*

The AFTA scheduler supports the execution of event-triggered hard real-time aperiodic tasks by statically assigning the processing associated with each given event with an RG. An appropriate RG is determined a priori by the maximum allowable time between the occurrence of the event and the VG's output response. Events may of course occur at any time. The AFTA Input/Output System Service (IOSS) is scheduled at the beginning of each frame and is responsible for reading the status of any events to be processed in subsequent frames. Thus there is at most one minor frame's latency between the time of an event's occurrence and the time at which the IOSS processes the event for delivery to the destination task. An event processing task may be assigned to rate groups 1 through 4, in some cases preempting iterative tasks as outlined below. Multiple event processing tasks may be scheduled on a VG. The following table illustrates the maximum event response time as a function of the RG containing the event processing task.

Rate Group	Maximum Event Response Latency, # Minor Frames	Maximum Event Response Latency, ms (10 ms Minor Frame)
1	16	160
2	8	80
3	4	40
4	2	20

Table 3-1. Maximum Event Response Latency vs. Rate Group

Figure 3-54 shows an event occurring in frame 0 of a rate group schedule. If the event processing task is in R4, then the response from the event is delivered at the end of frame 1. If the task is in R3, the response is delivered at the end of frame 3. If the task is in R2, the response is delivered at the end of frame 7, and if the task is in R1, the response is delivered at the end of frame 7 of the subsequent major frame (not shown in the figure).

Hard real-time event processing tasks are assigned to Rate Groups and are scheduled based on the occurrence of the events they are designed to handle. The arrival of a high-priority event and the consequent scheduling of an event processing task may perturb the timing of periodic tasks. Several options exist for scheduling event-triggered hard real-time aperiodic tasks.

One may validate the task suite's execution time upper-bound in the presence of all "valid" event combinations. The advantage of this approach is predictability and validatability for foreseen event suites. The programmer need not worry about frame slippage due to event-based preemption. The disadvantages are potential poor processor utilization, undefined or unpredictable behavior should an unforeseen event suite occur, and lengthy validation.

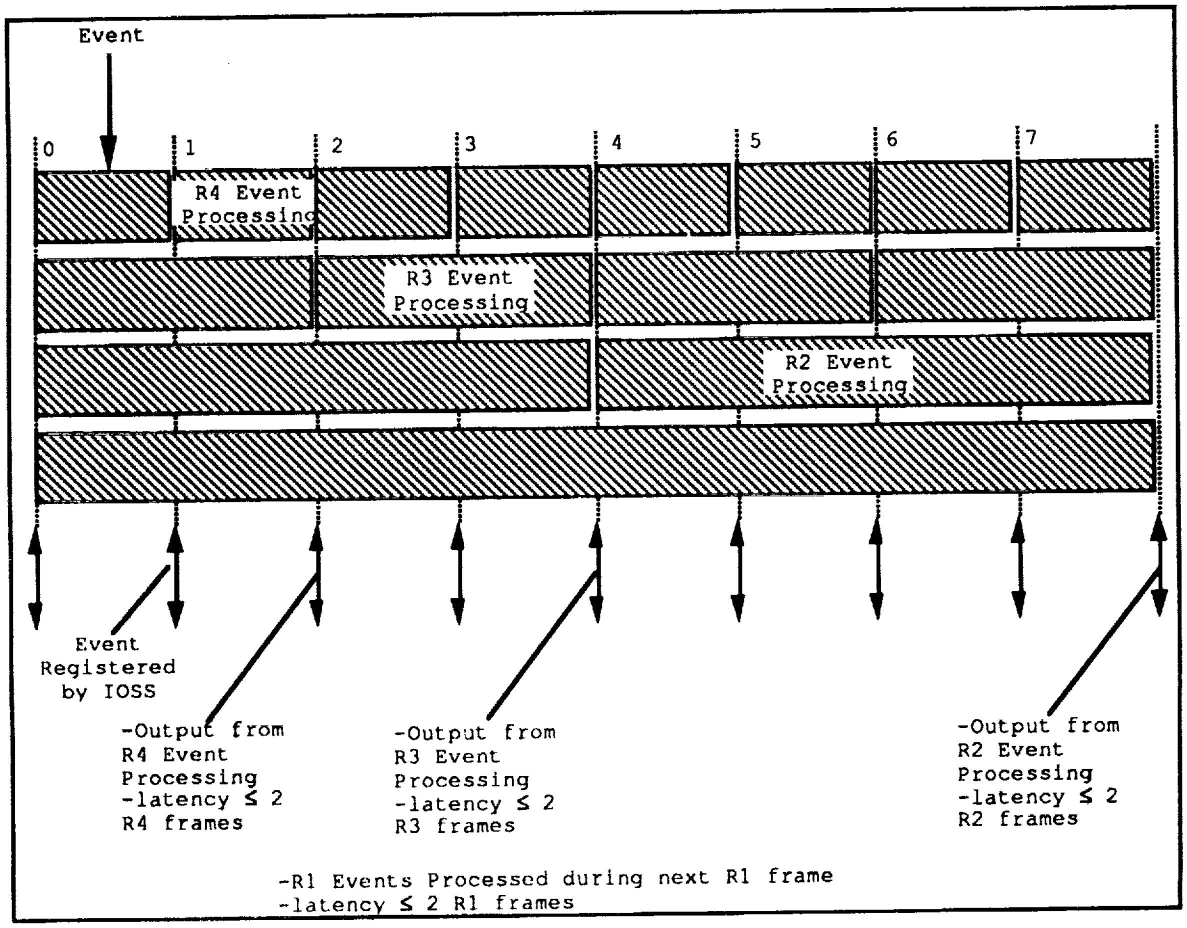

Event
0
1
2
3
4
5
6
7
R4 Event Processing
R3 Event Processing
R2 Event Processing
Event Registered by IOSS
-Output from R4 Event Processing -latency ≤ 2 R4 frames
-Output from R3 Event Processing -latency ≤ 2 R3 frames
-Output from R2 Event Processing -latency ≤ 2 R2 frames
-R1 Events Processed during next R1 frame -latency ≤ 2 R1 frames

Figure 3-54. Scheduling of Event-Triggered Hard Real-Time Aperiodic Tasks

Alternatively, depending on the event to be processed, one can deschedule one or more selected periodic tasks of equal *or higher* iteration rate. This can be done by the RG dispatcher function. For example, an event which must be processed in an R3 frame (i.e., a maximum response time of 40 ms) could cause an R4 task to be descheduled, or it could cause an R3 task of higher precedence order to be descheduled. These preemption trades *must* be selected beforehand at the time the application designer defines the relative importance and urgencies of the various conditions facing the computing system, rather than waiting until system integration time to perform empirical determinations of whether the real-time responses to external events are met, followed by adjusting task priorities accordingly. The effect of frame slippage on the descheduled tasks must be defined and tolerance means must be developed. After event processing completion, the descheduled iterative tasks must be rescheduled for resumption.

It is critical that, regardless of the selected option, periodic and aperiodic hard real-time task aggregate execution times must be validated to meet all real time constraints.

3.5.1.4.5. *Aperiodic Non-Real-Time Task Scheduling*

Aperiodic tasks which do not have hard real-time constraints are executed after all rate group tasks (including aperiodic hard real-time tasks) have been executed. There may be several non-real-time aperiodic tasks running on a VG and they may be scheduled arbitrarily (unprioritized round-robin, multi-level prioritized, etc.). Messages and I/O operations emanating from aperiodic tasks are handled differently from those emanating from RG tasks. When an aperiodic task wishes to transmit a message or execute an I/O operation, the message is appended to an asynchronous queue specific to that aperiodic task. On every frame, the dispatcher task performs a series of Class 2 Exchanges regarding the status of this queue to determine if all copies of the aperiodic task have requested the message transmission, or if a majority of the copies have requested the transmission and a suitably long timeout interval has expired such that the minority of the copies can be presumed to be faulty. If either of these conditions are met, the dispatcher transmits the message.

The aperiodic task timeout calculation is based on the amount of processor time the task consumed since its last synchronization act. This quantity defines the elapsed execution time of the task. All copies of the task may not have identical elapsed execution times, so a fault tolerant algorithm must be employed to exchange and agree upon a valid execution time. Given that all copies of the dispatcher agree upon a valid elapsed execution time, the timeout may be calculated if the amount of skew buildup per processor time unit is known.

3.5.1.4.6. *Execution of RGs on Multiple VGs*

Due to the parallel nature of AFTA, different VGs will execute different RG task suites. Mapping a multi-VG multi-RG task suite onto multiple VGs can be performed using application task-to-parallel processor mapping technology embodied in an integrated schedule generation and analysis tool. Task suites are expected to change as a function of the mission mode and system state. This will give rise to multiple mappings. Each such mapping must be created using the schedule generation and analysis tool. Moreover, the valid transition sequences from one such mapping to another must be carefully defined and implemented so as to continue to meet real-time requirements during the transition period.

The rate group *phasing* describes the relationship between the rate group frames on different VGs in the system. Within the task configuration table, each task is assigned to execute in some rate group. The rate group determines the frequency at which the task will be executed and the resulting rate group frame delimits the execution cycle of the task. Tasks assigned to the same rate group will execute at the same frequency regardless of their hosting VG, but there may be a time difference between the start of their first and each subsequent rate group frame if the tasks are executing on different VGs. This phasing would be caused by the completion of system initialization at different times on different VGs. An example phasing of the frames for tasks in a given rate group on multiple VGs is shown in Figure 3-55.

In the example, the first rate group frame on VG1 starts at the base time and the first rate group frames on the remaining VGs are delayed. The interval between the base time and the start of the first rate group frame is the VG's phase delay. The phase delay is important because it determines the relationship between the frame in which messages are sent and the frame in which they are received. It also effects the degree to which the different VGs contend for the Network Elements and other physical resources. This is also affected by the message passing restrictions in the rate group tasking paradigm. In the paradigm, a task's queued messages are only sent and its received messages are only made available at its corresponding rate group frame boundary. This is indicated in the figure by the arrows at the frame boundaries. An example from the figure is the messages transmitted after the first frame on VG1. They will be received at the start of the first frame on VG2 and VG4, but will not be received until the start of the second frame on VG3. This relationship of sending frame to receiving frame will remain constant for subsequent frames if the phasing does not change.

Unfortunately, the phasing will change if the start of subsequent rate group frames on different VGs are allowed to float with respect to each other. The time management service (embodied in the rate group dispatcher task) has been designed to minimize this float by locking the phase to the system time maintained by the Network Element. There still remains inherent float because of the variability of the interval from the start of the frame to when any given message will be sent or read. This float is increased when VGs which share a Network Element have the same phase delay or their delays differ by an integer number of minor frames. This is because the VGs are then forced to compete for access to the Network Element to send and read their messages at their frame boundary. For this reason the simplest phasing of a zero phase delay for all VGs is not recommended. The phase field in the VG configuration table is provided to specify the desired phase delay for each VG.

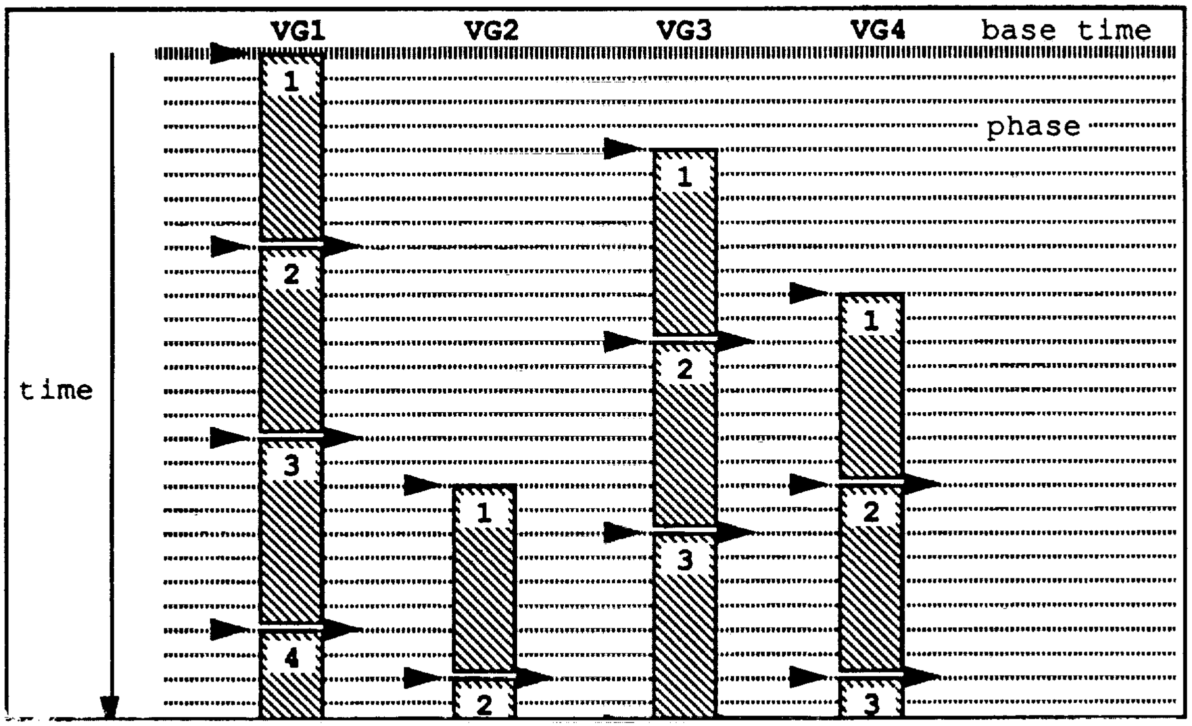


Figure 3-55. Phasing of RG Frames on Multiple VGs

3.5.2. Fault Detection, Identification and Recovery Overview

AFTA uses hardware redundancy with fault detection and masking capabilities to provide fault tolerance. While the hardware alone in AFTA could sustain a single fault, the Fault Detection, Identification and Recovery (FDIR) software allows it to sustain multiple successive faults by identifying a faulty component and eliminating that faulty component from the AFTA system operation, thereby restoring AFTA to its original fault tolerance capability.

AFTA achieves its reliability by testing itself through various means and by monitoring inherent fault detection mechanisms such as the NEs' message voters. When a fault is detected and isolated to a specific component, that faulty component is disabled in order to maintain a system of high reliability. Because reliability of the system is of utmost importance, AFTA is thoroughly tested to eliminate faulty components from the operational system. This testing occurs at all stages of AFTA's operations. As the computing system proceeds through the various operating modes from an initial power on state to a fully operational mode, the testing methodology also evolves through various modes of testing commensurate with the operational constraints. During each testing mode, suites of tests are activated to exercise all AFTA components both individually and systematically as comprehensively as possible. Specifically, there are 3 test modes – initial power-on test mode (I-BIT), line maintenance test mode (M-BIT), and continuous test mode (C-BIT) – and 3 system modes – power on, standby, and mission operational. Figure 3-56 depicts the interaction of the system and test modes.

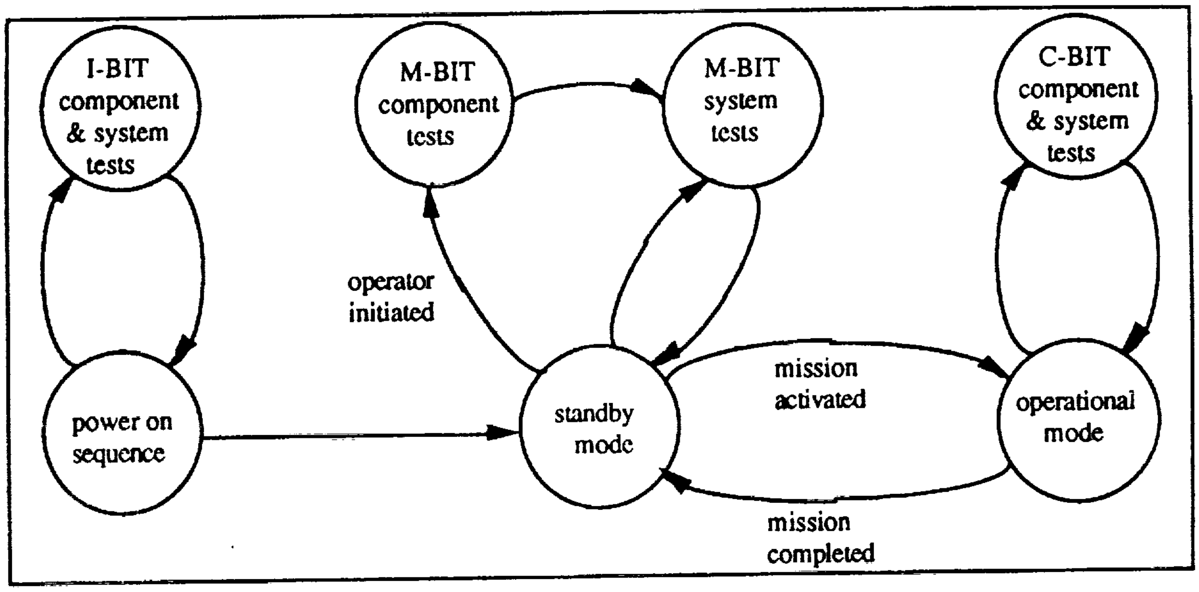


Figure 3-56. System mode and test mode interactions

During each of these test modes both component level tests and system level tests are executed. The component level tests exercise the individual AFTA components whereas the system level tests employ the inherent fault detection capabilities and the operating characteristics of AFTA to identify faulty components. However, during each system mode, the time constraints, the requirements on maintenance of mission critical information, and the system configurations differ. Consequently, the tests executed during each of the test modes vary based upon these factors. The constraints on these test modes and the delineation of the tests executed during the various test modes are discussed extensively in Section 5.6.

The fault detection, isolation and recovery task is responsible for testing of all components of AFTA during all system modes. It tests all AFTA components during the power on sequence and invokes a more comprehensive set of tests when directed by a line maintenance operator. During mission critical operational mode it continuously monitors fault detection mechanisms and performs self testing.

When a faulty component has been identified, FDIR initiates an appropriate recovery strategy which attempts to compensate for the loss of a component. The variety of recovery strategies is numerous not only because the policy must be commensurate with the type of component failed but also because of the system requirements and the mode of operation. The array of recovery policies includes a strategy to replace a faulty Processing Element with a spare Processing Element, an option to migrate a task when its Processing Element fails, and a policy to quickly mask the incorrect behavior of a failed component.

As faulty components are identified and eliminated from the operational AFTA system, FDIR maintains logs of these faults to aid in the task of maintaining the AFTA hardware. These logs identify the line replaceable module deemed faulty and provide detailed diagnostic information, to the chip level whenever possible. The FDIR task reports this information to line maintenance personnel upon request.

3.5.3. Input/Output Services

AFTA interfaces to a wide variety of inputs and outputs. AFTA I/O is accessed through I/O Controllers (IOCs) which, in the Brassboard, reside on the FCR backplane bus connecting the PEs and the NEs. IOCs can either be simplex or redundant. They may also either be members of a specially-designated I/O VG or may be controlled over the FCR backplane bus (or auxiliary bus) by one or more designated PEs.

Desired characteristics of the AFTA I/O process are that the load modules of different members of a redundant VG must be identical, even if only a subset of the members actually execute the I/O operation. The second requirement is that the control flows of redundant VGs executing I/O be similar if not identical, even if only a subset of the members actually execute the I/O operation; heterogeneous I/O must not be allowed to induce sufficient skew to force the desynchronization of a redundant VG. Finally, when redundant I/O is accessed, it is important that the copies of the I/O device be accessed at very close to the same time. To minimize jitter, it is planned that all I/O activity will be synchronized with frame boundaries.

3.6. The Role of Standards in the AFTA

The validity and applicability of the AFTA architectural concept is intended to be independent of prescriptive engineering standards which may apply to various applications. The ability to comply with differing standard suites without belying the validity of the AFTA architectural concept is indeed one of the strong points of the AFTA architecture, and is intended to give it a degree of universal applicability.

A given AFTA implementation is constructed under due compliance with the standards asserted by the procuring agency. Standards may exist singly or may be aggregated into suites, and may apply to commercial, military, space, and other applications. For the purposes of AFTA, we are primarily interested in existing or emerging standards which apply to military applications. Standard suites may include backplane buses, Instruction Set Architectures (ISAs), programming languages, interconnection network hardware and topologies, physical dimensions, connectors, communication protocol stacks, operating system services, and development, testing, and documentation procedures.

Standards serve several valuable purposes. They assist in fixing unknown design parameters, thus removing many uncertainties regarding design. This in turn allows educated estimations of critical system parameters at an early enough stage in the design to detect and rectify potential problems. The existence of standards implies the existence of large body of standard AFTA building blocks such as processors, input/output devices, interfaces, and software. This results in the potential for low prototyping and procurement costs, and the components' maturity reduces the probability of design flaws. The availability of mature standards-compliant components substantially reduces the risk, schedule, and cost involved in utilizing them in AFTA. As higher performance, lower cost, more reliable, etc. standards-compliant components are developed, AFTA can be upgraded, thus leveraging advances in microelectronics technology.

3.7. Relationship of AFTA to the Advanced Information Processing System

Over the past few years NASA and the Strategic Defense Initiative Office (SDIO) have sponsored the Advanced Information Processing System (AIPS) program at Draper Laboratory. The overall goal of the AIPS program is to produce the knowledgebase necessary to achieve validated distributed fault tolerant computer system architectures for advanced real-time aerospace applications [Har91b]. As a part of this effort, an AIPS engineering model consisting of hardware building blocks such as Fault Tolerant Processors and Inter-Computer (IC) and Input/Output (I/O) networks and software building blocks such as Local System Services, IC and I/O Communications Services was constructed. Figure 3-57 shows the laboratory configuration of the engineering model. It consists of three triplex FTPs and a simplex processor interconnected by a triplex IC network. The IC network was designed with 5 nodes in each layer. Each node services one processing site. One of the nodes in each layer was left unused for expandability. Figure 3-58 shows how the FTPP, which can be considered one of the AIPS building blocks, can be interfaced to the AIPS IC network. A node from each of the three IC layers would interface with an ICIS-like I/O controller in a different fault containment region of the FTPP. The I/O controller will be designed to participate in the modified Laning Poll contention scheme used to arbitrate access to the AIPS IC network.

As an illustration of the use of the AIPS building blocks, including the FTPP, for an advanced aerospace application, consider the Advanced Launch System. The AIPS building blocks and the knowledgebase were used to synthesize an avionics architecture for the Advanced Launch System (ALS) [Lal91]. The architecture consisted of a quadruply redundant bi-processor (two processors per channel) for the core avionics functions and a triplex processor for each engine, all networked together by the IC network. With the availability of the FTPP, the core avionics FTP can be replaced with an FTPP consisting of four Network Elements each of which services two processors. This will provide the required reliability, availability and throughput but, unlike the FTP, can be built out of NDI components.

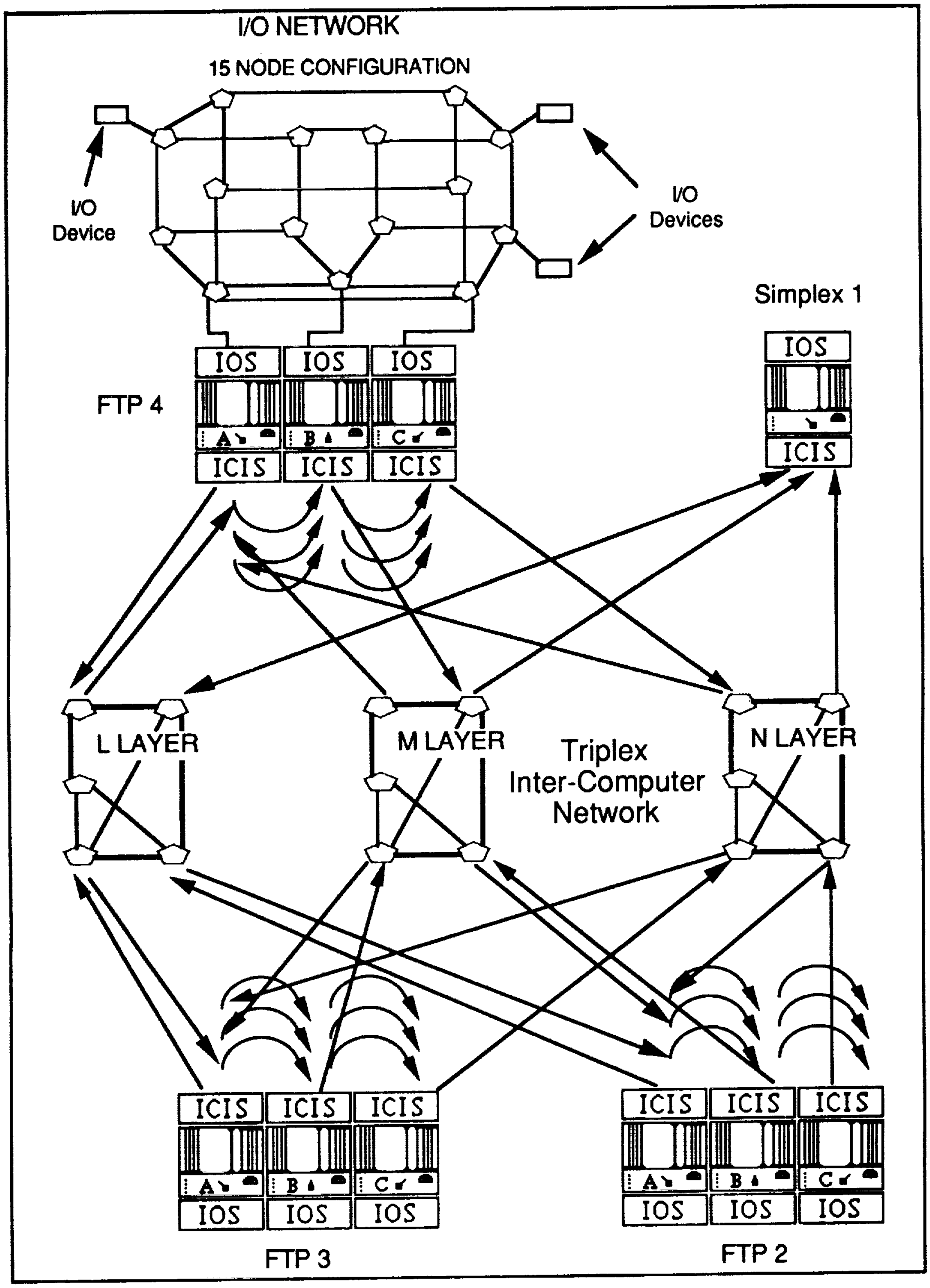


Figure 3-57. The Advanced Information Processing System Engineering Model

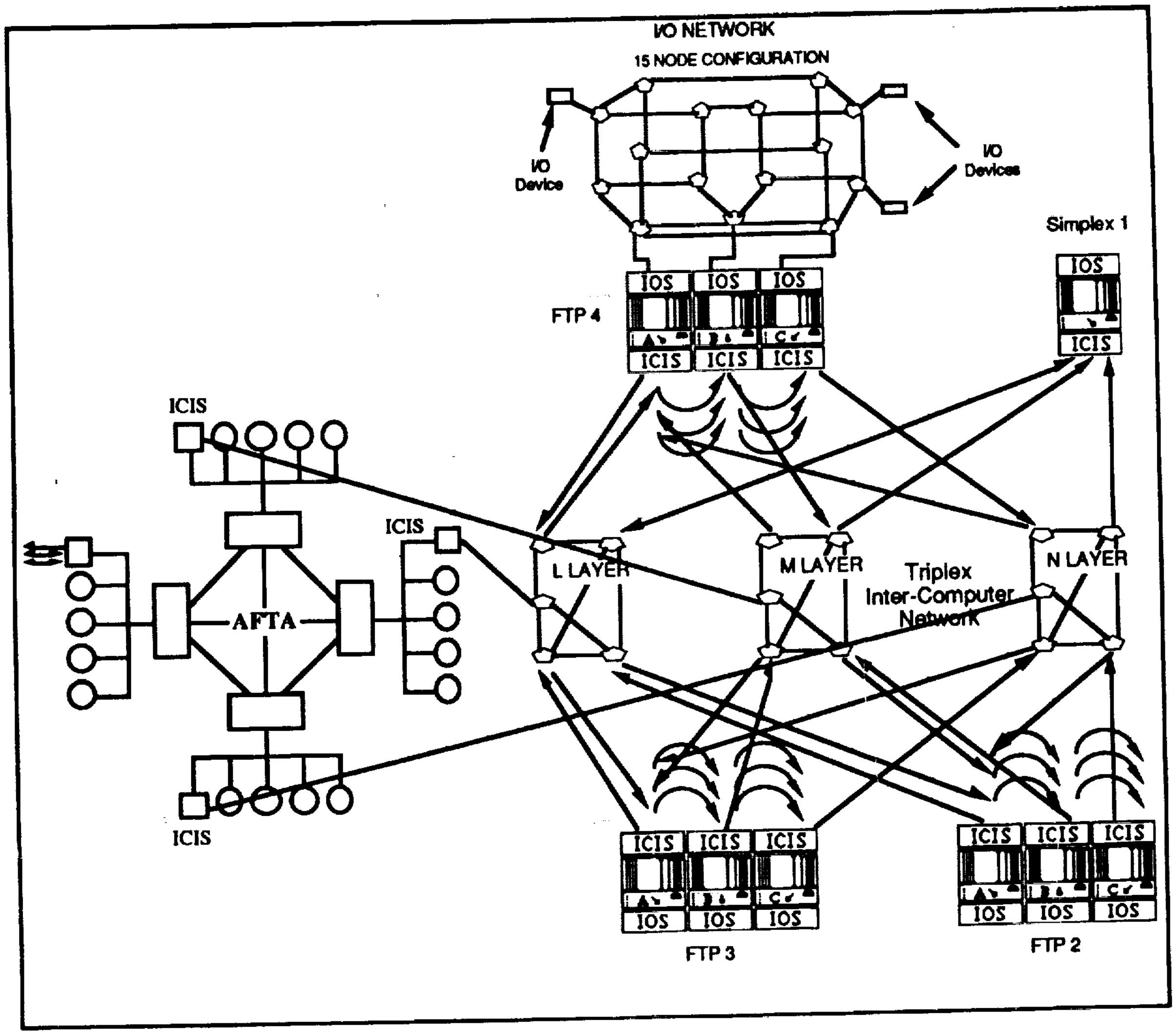


Figure 3-58. Interface Between AFTA and the Advanced Information Processing System Engineering Model

This page intentionally left blank.

Appendix A. References

[Abl88]	Abler, T., *A Network Element Based Fault Tolerant Processor*, MS Thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1988.
[AMD89a]	*The SUPERNET Family for FDDI*, Advanced Micro Devices Data Book, Publication # 09734 Rev. C, February 1989.
[AMD89b]	*Am7968/Am7969-175 TAXIchip^{TM} Integrated Circuits*, Advanced Micro Devices Data Sheet, Publication # 12834 Rev. A, November 1989.
[ANSI139]	"Fiber Distributed Data Interface (FDDI) - Token Ring Media Access Control (MAC)" American National Standard, ANSI X3.139-1987, November 5, 1986.
[ANSI148]	"Fiber Distributed Data Interface (FDDI) - Token Ring Physical Layer Protocol (PHY)," American National Standard, ANSI X3.148-1988, June 30, 1988.
[ANSI166]	"Fibre Data Distributed Interface (FDDI) - Token Ring Physical Layer Medium Dependent (PMD)," American National Standard, ANSI X3.166-1990, September 28, 1989.
[APS90]	Acarlar, M. S., Plourde, J. K., Snodgrass, M. L., "A High Speed Surface-Mount Optical Data Link for Military Applications," IEEE/AIAA/NASA 9th Digital Avionics Systems Conference Proceedings, October 15-18, 1990, p. 297-302.
[Bab90]	Babikyan, C., "The Fault Tolerant Parallel Processor Operating System Concepts and Performance Measurement Overview," *Proceedings of the 9th Digital Avionics Systems Conference,* October 1990, pp. 366-371.
[Ber87]	Bertsekas, D., Gallager, R., Data Networks, Prentice-Hall, 1987.
[Ber90]	Berger, K. M., Abramson, M. R., Deutsch, O. L., "Far-Field Mission Planning for Helicopters," CSDL Technical Report CSDL-R-2234, March 1990.
[Bev90]	Bevier, W.R., and Young, W.D., "The Proof of Correctness of a Fault-Tolerant Circuit Design," 2nd International Working Conference on Dependable Computing for Critical Applications, Tucson, AZ, February 1991.
[Bic90]	Bickford, M., and Srivas, M., "Verifying an Interactive Consistency Circuit: A Case Study in the Reuse of a Verification Technology," NASA Formal Methods Workshop 1990, NASA Conference Publication 10052, November 1990.

[Biv88] Bivens, G. A., "Reliability Assessment of Surface Mount Technology (SMT)," RADC report RADC-TR-88-72, March 1988.

[Bla91] Black, Uyless, OSI : A Model For Computer Communications Standards, Prentice-Hall, 1991.

[Boo88] Booth, F., "Advanced Apache Architecture," 8th Digital Avionics Systems Conference, October 1988.

[Bur89] Burkhardt, L., *Advanced Information Processing System: Local System Services*, NASA Contractor Report 181767, April 1989.

[But88] Butler, R. W., "A Survey of Provably Correct Fault Tolerant Clock Synchronization Techniques," NASA Technical Report TM-100553, NASA Langley Research Center, February 1988.

[CAMP] CAMP-1 Final Technical Report AFATL-TR-85-93, 3 Volumes, Available as DTIC AD-B102 654, AD-B102 655, and AD-B102 656 from Defense Technical Information Center, Alexandria, VA 22304-6145.

[Car84] Carlow, G. D., "Architecture of the Space Shuttle Primary Avionics Software System", *Communications of the ACM*, 27(9):926-36, September 1984.

[Cha84] Chambers, F. B., ed., *Distributed Computing*, Academic Press, 1984.

[Che87] Cheng, S-C., Stankovic, J. A., Ramamritham, K., "Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey," in Hard Real-Time Systems, *IEEE Computer Society Press*, 1988.

[Coh87] Cohn, Marc D., "The Conformance of the ANSI FDDI Standard to the SAE-9B HART High Speed Data Bus Requirements for Real-Time Local Area Networks," Society of Automotive Engineers Aerospace Systems Conference Proceedings, November 1987.

[Coh88] Cohn, Marc D., "The Fiber Optic Data Distribution Network: A Network for Next-Generation Avionics Systems," AIAA/IEEE 8th Digital Avionics Systems Conference Proceedings, October 17-20, 1988, p. 731-737.

[Coh90a] Cohen, G. C., et. al., *Design of an Integrated Airframe/Propulsion Control System Architecture*" NASA Contractor Report 182004, March 1990.

[Coh90b] Cohen, G. C., et. al., *Final Report: Design of an Integrated Airframe/Propulsion Control System Architecture*, NASA Contractor Report 182007, March 1990.

[Cohn88] Cohn, A., "Correctness Properties of the Viper Block Model: The Second Level," Tech. Report 134, Univ. of Cmabridge, Cambridge, England, May 1988.

[Com91]	Comer, D. E., Internetworking with TCP/IP, Prentice-Hall, 1991.
[CSDL9214]	Completion of the Advanced Information Processing System, response to NASA Langley Research Center, CBD Announcement REF SS017, issue PSA-9214, November 12, 1986.
[Cullyer 88]	Cullyer, W. J., "Implementing Safety-Critical Systems: The VIPER Microprocessor," VLSI Specification, Verification and Synthesis, Kluwer Academic Publishers, 1988.
[CVC2]	"System Specification for Combat Vehicle Command and Control (DRAFT)," CVC2 Systems Implementation Working Group, 31 October 1990.
[DACS]	Defense & Analysis Center for Software, Kaman Sciences Corporation, P.O. Box 120, Utica, NY 13503.
[Dal73]	Daly, W.M., Hopkins, A.L., and McKenna, J.F., "A Fault-Tolerant Digital Clocking System," 3rd International Symposium on Fault Tolerant Computing, Palo Alto, CA, June 1973.
[Deu88]	Deutsch, O. L., Desai, M., "Development and Demonstration of an On-Board Mission Planner for Helicopters," CSDL Technical Report CSDL-R-2056, April 1988.
[DID80811]	"VHSIC Hardware Description Language (VHDL) Documentation," Data Item Description, DD Form 1664, DI-EGDS-80811, May 11, 1989.
[DiV90]	Di Vito, B. L., Butler, R. W., Caldwell, J. L., *Formal Design and Verification of a Reliable Computing Platform for Real-Time Control*, NASA Technical Memorandum 102716, October 1990.
[DiV91]	Di Vito, B., Butler, R., and Caldwell, J., "High Level Design Proof of a Reliable Computing Platform," 2nd International Working Conference on Dependable Computing for Critical Applications, Tucson, AZ, February 1991.
[Dol82]	Dolev, D., "The Byzantine Generals Strike Again," *Journal of Algorithms*, Vol. 3, 1982, pp. 14-30.
[Dol84]	Dolev, D., Dwork, C., Stockmeyer, L., "On the Minimal Synchronism Needed for Distributed Consensus," IBM Research Report RJ 4292 (46990), 5/8/84.
[Fel90]	Felter, S. C., Douglas, P. H., Smith, C. A., "Avionics System Integration for the MH-53J Helicopter," 9th Digital Avionics Systems Conference, October 1990.
[Fis82]	Fischer, M. J., Lynch, N. A., "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, Vol. 14, No. 4, 13 June 1982, pp. 183-186.

[Foh89]	Fohler, G., Koza, C., "Heuristic Scheduling for Distributed Real-Time Systems," Research Report No. 6/89, Institut fur Technische Informatik, Technische Universitat Wien, Vienna, Austria, April 1989.
[Gal90]	Galetti, R. R., Real-Time Digital Signatures and Authentication Protocols, Master of Science thesis, Massachusetts Institute of Technology, May 1990.
[Goe91]	Goel, A.L., and Sahoo, S.N., "Formal Specifications and Reliability: An Experimental Study," 1991 International Symposium on Software Reliability Engineering, Austin, Texas, May 1991.
[Gua90]	Guaspari, D., Marceau, C., and Polak, W., "Formal Verification of Ada Programs," IEEE Transactions on Software Engineering, Special Issue on Formal Methods in Software Engineering, Vol. 16, No. 9, September 1990.
[Han89]	Hanaway, J. F., Morrehead, R. W., *Space Shuttle Avionics System*, NASA SP-504, 1989.
[Har87]	Harper, R., *Critical Issues in Ultra-Reliable Parallel Processing*, PhD Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1987.
[Har88a]	Harper, R., Lala, J., Deyst, J., "Fault Tolerant Parallel Processor Overview," *18th International Symposium on Fault Tolerant Computing*, June 1988, pp. 252-257.
[Har88b]	Harper, R., "Reliability Analysis of Parallel Processing Systems," *Proceedings of the 8th Digital Avionics Systems Conference.*, October 1988, pp. 213-219.
[Har91a]	Harper, R., Lala, J., *Fault Tolerant Parallel Processor*, J. Guidance, Control, and Dynamics, V. 14, N. 3, May-June 1991, pp. 554-563.
[Har91b]	Harper, R., Alger, L., Lala, J., "Advanced Information Processing System: Design and Validation Knowledgebase," NASA Contractor Report 187544, September 1991.
[Hir90]	Hird, G.R., "Formal Methods in Software Engineering," 9th AIAA/IEEE Digital Avionics Systems Conference, Virginia Beach, VA, October 1990, pp. 230-234.
[Hun86]	Hunt, W.A., "FM8501: A Verified Microprocessor," Proceedings of IFIP Working Group 10.2 Workshop, North Holland, Amsterdam, 1986.
[Hwa84]	Hwang, K., Briggs, F., <u>Computer Architecture and Parallel Processing</u>, McGraw-Hill, 1984.
[IEEE1076]	"VHDL Language Reference Manual," IEEE Standard, IEEE Std 1076-1987, March 31, 1988.

[IEEE8021]	"Local and Metropolitan Area Networks: Overview and Architecture," IEEE Standard, IEEE Std 802-1990, May 31, 1990.
[IEEE8022]	"Logical Link Control," IEEE Standard, IEEE Std 802.2-1989, August 17, 1989.
[IEEE8023]	"Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," IEEE Standard, IEEE 802.3-1988, June 9, 1988.
[IEEE8024]	"Token-Passing Bus Access Method and Physical Layer Specifications," IEEE Standard, IEEE 802.4-1990.
[J88N2]	"Linear Token Passing Multiplex Data Bus Protocol," Joint Integrated Avionics Working Group Standard, Document J88-N2,
[J8701]	"Advanced Avionics Architecture (A3) Standard," Joint Integrated Avionics Working Group Standard, Document J87-01.
[Klj89]	Kljaich, J., Jr., Smith, B.T., and Wojcik, A.S., "Formal Verification of Fault Tolerance Using Theorem-Proving Techniques," IEEE Transactions on Computers, Vol. 38, No. 3, March 1989.
[Kop89]	Kopetz, H., et. al., "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach," *IEEE Micro*, 9(1):25-40, February 1991.
[Kop91]	Kopetz, H., et. al., "The Rolling Ball on MARS," Institut fur Technische Informatik Research Report No. 13/91, Technische Universitat Wien, Vienna, Austria, November 1991.
[Kri85]	Krishna, C. M., Shin, K. G., Butler, R. W., "Ensuring Fault Tolerance of Phase Locked Clocks," *IEEE Trans. Computers*, Vol. C-34, No. 8, August, 1985.
[Lal84]	Lala, J. H., "An Advanced Information Processing System," 6th AIAA-IEEE Digital Avionics Systems Conference, Baltimore, MD, Dec. 1984.
[Lal84]	Lala, J. H., "An Advanced Information Processing System," 6th AIAA-IEEE Digital Avionics Systems Conference, Baltimore, MD, December 1984.
[Lal85]	Lal·., J. H., "Advanced Information Processing System: Fault Detection and Error Handling," AIAA Guidance, Navigation and Control Conf., Snowmass, CO, Aug. 1985.
[Lal86a]	Lala, J.H., "Fault Detection, Isolation, and Reconfiguration in the Fault Tolerant Multiprocessor," Journal of Guidance, Control, and Dynamics, Sept-Oct. 1986.
[Lal86b]	Lala, J. H., "A Byzantine Resilient Fault Tolerant Computer for Nuclear Power Plant Applications," 16th Annual International Sym-

posium on Fault Tolerant Computing Systems, Vienna, Austria, 1-4 July 1986.

[Lal89] Lala, J.H., et. al., "Study of a Unified Hardware and Software Fault Tolerant Architecture," NASA Contractor Report 181759, January 1989.

[Lal91] Lala, J.H., R. Harper, K. Jaskowiak, G. Rosch, L. Alger, and A. Schor "AIPS for Advanced Launch System: Architecture Synthesis Report", NASA Contractor Report 187544, September 1991.

[Lam85] Lamport, L., Melliar-Smith, P. M., "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM*, 32(1):52-78, January 1985.

[Lap90] "Dependability: Basic Concepts and Terminology," J.C. Laprie - Editor, Published by International Federation for Information Processing (IFIP) Working Group 10.4 on Dependable Computing and Fault Tolerance, December 1990.

[Leh87] Lehoczky, Sha, Ding, *The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behavior*, Technical Report, Department of Statistics, Carnegie-Mellon University, 1987.

[Liu73] Liu, C. L., Layland, J. W., "Scheduling Algorithms for Multiprograming in a hard Real-time Environment," *J. ACM*, 20(1):46-61, 1973.

[LSP82] Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem," ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, p. 382-401.

[MA-HDBK] Modular Avionics Handbook, Document No. 21530(0-6), FSCM 51993, Draft C, U. S. Air Force ASD-ALD/AX, 19 April 1990.

[Ma78] Martin, D. L., Gangsaas, D., "Testing of the YC-14 Flight Control System Software," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 1, No. 4, July-August 1978.

[McE88] McElvany, M. C., "Guaranteeing Deadlines in MAFT," *IEEE Real-Time Systems Symposium*, Huntsville, AL, December 1988.

[MIL-HDBK-0036] "Survivable Adaptable Fiber Optic Embedded Network II - SAFENET II," Military Handbook, MIL-HDBK-0036, 1 March, 1990.

[MIL-HDBK-59] MIL-HDBK-59, "Computer-Aided Acquisition and Logistic Support (CALS) Program Implementation Guide," 20 December 1988.

[MIL-HDBK-217E] MIL-HDBK-217E, "Reliability Prediction of Electronic Equipment," 2 January 1990.

[MIL-STD-344] MIL-STD-344 (draft), "Standard Army Vetronics Architecture," 14 September, 1990.

[MIL-STD-785B]	MIL-STD-785B, "Reliability Program for Systems and Equipment Development and Production," 15 September 1980.
[MIL-STD-1553]	"Aircraft Internal Time Division Command/Response Multiplex Data Bus," Military Standard, MIL-STD-1553B, 12 February, 1980.
[MIL-STD-1815A]	MIL-STD-1815A, "Reference Manual for the Ada Programming Language," 17 February 1983.
[NAS1-18565-14]	Statement of Work for NASA Contract NAS1-18565, Task 14, June 1990.
[Osd88]	Osder, S. S., "Digital Fly-by-Wire System for Advanced AH-64 Helicopters," 8th Digital Avionics Systems Conference, October 1988.
[Pe80]	Pease, M., Shostak, R., Lamport, L., "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, No. 2, April 1980, pp. 228-234.
[PEI90120]	*XTP® Protocol Definition, Revision 3.5*, Published by Protocol Engines Inc., September 1990.
[Pek88]	Pekelsma, N. J., "Optimal Guidance with Obstacle Avoidance for Nap-of-the Earth Flight," NASA Contractor Report 177515, December 1988.
[Pus89]	Puschner, P., Koza, C., "Calculating the Maximum Execution Time of Real-Time Programs," *Real-Time Systems*, 1(2):159-176, September 1989.
[Rad90]	PMV 68 CPU-3A Specification, Issue 3, Publication No. 681/SA/04085, Radstone Technology plc, 1990.
[Rus89]	Rushby, J., von Henke, F., "Formal Verification of a Fault Tolerant Clock Synchronization Algorithm," NASA Contractor Report 4239, June 1989.
[SAE91]	SAE/AS-2A Subcommittee RTMT Statement on Requirements for Real-Time Communication Protocols (RTCP), Issue #1, SAE ARD50007, August 2 1991.
[San90]	STAR MVP Technical Description, Document No. 4069718, Lockheed Sanders, 25 June 1990.
[Sch91]	Schutz, W., "On the Testability of Distributed Real-Time Systems," *Proc. Tenth Symposium on Reliable Distributed Systems*, Pisa, Italy, September, 1991.
[Spi89]	Spivey, J.M., The Z Notation, A Reference Manual, Prentice Hall International (UK) Ltd, 1989.
[Spi90]	Spivey, J.M., "Specifying a Real-Time Kernel," IEEE Software, Special Issue on Formal Methods, Vol. 7, No. 5, Sep 1990.

[Sri90] Srivas, M. and Bickford, M., "Formal Verification of a Pipelined Microprocessor," IEEE Software, Special Issue on Formal Methods, Vol. 7, No. 5, September 1990.

[Sta87] Stankovic, J. A., Ramamritham, K., "The Design of the Spring Kernel," *Proc. of the Real Time Systems Symposium*, December 1987.

[Sun74] Sundstrom, R. J., "On-Line Diagnosis of Sequential Systems," PhD Thesis, University of Michigan, 1974.

[Tan88] Tanenbaum, A. S., Computer Networks, second edition, Prentice-Hall, 1988.

[X3T95] "FDDI Station Management (SMT)," Preliminary Draft Proposed American National Standard, X3T9.5/84-49, Rev. 6.2, May 18, 1990.

Appendix B. Glossary of Terms and Acronyms

AFTA-Army Fault-Tolerant Architecture-A computer designed for both high reliability and high throughput. The AFTA is based on the FTPP architecture.

aperiodic tasks-A set of tasks whose iteration rates are unknown or undefined.

ASIC-Application Specific Integrated Circuit-A type of integrated circuit that can be custom designed by the hardware engineer so that it will perform a particular logic or processing function and at the same time save circuit board space and power consumption. The advent of VLSI design techniques has made ASICs a more flexible and practical option for hardware designers.

ATP-Authentication Protocol-A protocol utilized by the BRNP to sign outgoing packets and to test the authenticity of incoming packets.

ATPG-Automatic Test Pattern Generation-The generation of test vectors directly from a netlist for verification of device functionality. Test vectors from an ATPG program do not test the correct functionality of the device; they only test that the device is a correct implementation of the design as specified by the netlist.

behavioral VHDL is defined to be a VHDL architecture which uses any of the legal VHDL constructs, including those which do not correspond to possible hardware realizations of the description (i.e., pure behavioral may not be synthesizeable). A level of description that specifies a device functionally in terms of output reactions to input stimulus. A behavioral description can also specify the timing relationships of inputs to outputs.

BIT-Built In Test-This is an internal diagnostic testing system that is included as part of the AFTA design. There are three forms of the BIT-- I-BIT is the initial power-on test system, M-BIT is for maintenance testing, C-BIT is the continuous in-flight test system.

BRNP-Byzantine Resilient Network Protocol-A network layer protocol which implements the Byzantine Resilient Virtual Circuit in order to guarantee that all messages are delivered accurately.

broadcast addressing-A method of station addressing using an identifier that causes all stations to respond to the specified address.

bypass-The ability to effectively isolate a node from the network without disrupting the continuity of the network.

Byzantine Resilient-Capable of tolerating Byzantine faults. A Byzantine Resilient system is capable of handling arbitrarily malfunctioning components that may supply faulty information to other parts of the system thereby causing a spread of faulty information within the system.

C3-Cluster 3-An FTPP model number. Composed of either 4 or 5 FCRs, 3-40 processors, 1-40 VIDs, simplex, triplex, and quadruplex processor redundancy levels. Previous FTPP models were C1 (4 FCRs, 16 processors, 4-16 VIDs, simplex, duplex, triplex, and quadruplex processor redundancy levels) and C2 (4 FCRs, 4 processors, one fixed quad VID).

cache-A form of memory that is typically much faster and much smaller than main memory. Through utilization of cache memory, a processor's throughput will be increased. Typically cache memory acts as a staging area for data; information will be pulled from main memory and temporarily stored in cache while it undergoes processing.

CDU-Cockpit Display Unit-A cathode ray tube display located in the vehicle cockpit for display of system status. The CDU may display overall AFTA system status, LRU level status, or LRM level status.

CID-Communication Identification-A designation assigned to each task which is used for intertask communication.

class test-A test of the Network Element voting mechanism that requests a non-congruent message exchange selectively on each channel of a fault masking group.

cluster-An FTPP consisting of 4 or 5 FCRs containing at least one virtual processing site. Multiple clusters could be connected by a network device (such as a fault-tolerant data bus) to provide even greater throughput than a single cluster. Most references to an FTPP refer to a single cluster design.

CMF-Common Mode Fault-A type of malfunction which will cause multiple faults or complete execution failure within a redundant processing group. Common mode faults may result from software flaws, hardware bugs, design flaws, massive electrical upsets etc.

concurrent I/O-Input/Output processes that allow the associated virtual group to perform other tasks while I/O is collecting data. This allows for greater processor throughput.

CRC-Cyclic Redundancy Check-An error detecting code used in data communications that allows the unit receiving a message to ensure through binary mathematics that it is the same message sent by the transmitting unit.

CSMA/CD-Carrier Sense Multiple Access with Collision Detection-A form of media access control whereby a potential transmitting station will monitor the bus to ensure that it is clear before transmission begins. During transmission, the station also monitors the bus to check for message collisions. If a collision occurs, the message must be re-transmitted.

CT-Configuration Table-A table stored on the Network Element that contains the current configuration of the system, i.e. which processors are members of which virtual groups.

DAIS-Digital Avionics Instruction Set-A benchmark for measuring processor throughput.

depot test-A set of diagnostic level tests executed outside of the constraints of a real-time environment with emphasis on the isolation of chip level faults in these components. These tests would occur at a maintenance repair facility in contrast to the various forms of built-in testing.

DPRAM-Dual-Port Random Access Memory-The type of memory that occupies the data segment. It provides a buffer between the NE and the PE; both the NE and the PE may access the data segment asynchronously, provided that they do not attempt to access the same location.

DR-Discrepancy Report-A report that is filed whenever unexpected behavior of the hardware, software, or system is encountered. By recording observable symptoms of the system throughout testing, integration, verification and validation, one may better trace and identify system flaws.

entity-A specific instance of a protocol element in an Open Systems Interconnection layer or sublayer.

FCR-Fault Containment Region-Usually comprised of a number of line replaceable modules such as Processing Elements, Network Elements, input/output controller, and power conditioners. The AFTA is made up of four or five FCR's, and each FCR usually resides on a single circuit board (with the exception of the power conditioner). An interchangeable term for the FCR is Line Replaceable Unit or LRU.

FDDI-Fiber Distributed Data Interface-A networking standard developed by the American National Standards Institute to provide high bandwidth for Local Area Networks.

FDIR-Fault Detection, Identification and Recovery-FDIR software designed for the AFTA allows it to sustain multiple successive faults by identifying a faulty component and reconfiguring the AFTA system operation to compensate for the fault.

FIFO-First In First Out-A type of information buffer in which the data that is stored first chronologically will be the first to be extracted.

FMEA-Failure Modes and Effects Analysis

FMG-Fault Masking Group-A logical grouping of three or four processors to enhance the reliability of critical tasks. The members of an FMG execute the same code with the same data and periodically exchange messages to ensure that they produce the same outputs.

FTC-Fault Tolerant Clock-A distributed digital phase-locked loop used for synchronization of AFTA fault containment regions.

FTDB-Fault Tolerant Data Bus-A local area network designed around principles of Byzantine resilience. Its primary objective is to provide an optimal internetworking system between simplex and redundant processing sites.

FTNP-Fault Tolerant Navigation Processor-The initial ground vehicle application for the AFTA is for the navigations system in Armored Systems Modernization vehicles.

FTPP-Fault-Tolerant Parallel Processor-A computer designed for both high reliability and high throughput. The core of the FTPP is the Network Element.

functional reliability-The probability that a given function can be executed because its resources are operational.

functional synchronization-In maintaining synchronous operation, the members of a VID perform a synchronizing act after some sequence of functions has been completed. The sequence of functions between the synchronization points is referred to as a frame.

GC-Global Controller-A microcoded finite-state machine used to coordinate the functions throughout the Network Element.

graceful degradation-Through self-testing, a virtual group may identify a faulty member and gracefully degrade its redundancy level using a configuration table update message to eliminate the faulty channel.

IOC-Input/Output Controller-These devices connect the AFTA to the outside world, and they must be compatible with the bus connecting elements of the FCR. They may have a programmable processor on board to drive the I/O, or they may require off-board processors for operation.

IPS-Instructions Per Second-The number of machine language instructions that a processor will execute every second. This measurement is used to reference the speed of the processor.

ISO/OSI-International Standards Organization/Open Systems Interconnection-A specification and model for computer communication networks.

LAN-Local Area Network-A network topology that interconnects computer systems separated by relatively short distances (2-2000 meters). LAN technology is usually based on a shared medium with no intermediate switching nodes required.

leaf-level-(VHDL) The models at the bottom of the model tree. Leaf-level models in VHDL are always pure behavioral models.

LERP-Local Exchange Request Pattern-A string of bytes describing the current state of the input and output buffers for each processor in an FCR. The LERP is used to generate the SERP. Each FCR has a different configuration, therefore the LERPs for each FCR will be different. For this reason, LERPs must be treated as single-source data.

link-An element in a physical network that provides interconnection between nodes.

LOC-Loss of Control-This will occur as a result of a failure in any flight critical portion of the Flight Control System. For analysis purposes, LOC will be considered as a total loss of the vehicle.

Local FDI-Each virtual group will exercise its own fault detection and identification processes to monitor failures among its processors. Also, each virtual group may initiate its own recovery options.

logical addressing-A method of station addressing using an identifier that may select a group of stations to respond to the specified address.

LRM-Line Replaceable Module-The physical unit for field diagnosis and repair. Typically it consists of one circuit card assembly with one or more Processing Elements.

LTPB-Linear Token Passing Bus-A media access control method whereby stations pass a token along a virtual ring from one to another. A station may only transmit when it possesses the token.

MDC-Minimum Dispatch Complement-This specifies the absolute minimum level of operability for the AFTA system to be cleared for a sortie.

media access control-The method by which access to the physical network media is limited to a single node so that communications over the media are undisturbed.

media layer-One or more physical layer media. Multiple media layers are physically and electrically isolated from each other to the same degree as a fault-containment region in a fault-tolerant computer. Most traditional LANs use only a single network layer. A Byzantine resilient network usually employs multiple media layers for redundancy.

memory alignment-A process whereby the RAM and registers in each processor of a virtual group are made congruent as part of the resynchronization of a virtual group.

mission reliability-Arithmetically speaking, mission reliability is one minus the probability that failure of the AFTA causes abortion of the mission.

MMC-Minimum Mission Complement-This specifies the minimum level of AFTA operability for the vehicle to continue its mission.

NDI-Non-Developmental Item

NE-Network Element-The hardware device which provides the connectivity between virtual groups. The primary function of the NE is to exchange and vote packets of data provided by the processors. The ensemble of Network Elements forms a virtual bus network to which all virtual groups are connected.

NEID-Network Element ID-The name by which a Network Element is known in the physical AFTA configuration. An NEID refers to a specific Network Element in the system, i.e. the same NEID on different FCRs refers to the same Network Element. The NEID is also used to refer to the FCR in which the referenced Network Element resides. By convention, letters are used to denote the NEID.

netlist-A list defining interconnections of components. Netlists are typically used for designing printed circuit boards or ASICs.

NIU-Network Interface Unit-The connection between a station and the FTDB

node-An element in a physical network that provides the necessary interface between a station and the network media.

nonpreemptible I/O dispatcher-A task on the virtual group that manages the execution of certain I/O instructions that cannot be interrupted.

packet-A block of data consisting of a header, data, and a trailer exchanged between peer protocol entities. The term packet is somewhat generic and is applied at all levels of the protocol hierarchy.

packet-A string of data of fixed or variable length for transmission from one processor to another through an inter-processor network. A message-passing network handles data in packets. The term packet is used here to refer to a fixed-size (64 bytes) block of data which is transmitted by the Network Elements.

PDU-Protocol Data Unit-A fancy name for a packet. PDU is the name used by OSI.

PE-Processing Element-A hardware device which provides a general or special purpose processing site. A minimal PE configuration contains a single processor and local memory (RAM and ROM). PEs may optionally have private I/O, making them a combination PE and IOC.

PEID-Processing Element ID-The name by which a Processing Element is known in the physical AFTA configuration. Each PE in an FCR has a unique PEID. However, the same

PEID may be used by another processor in another FCR. A combination of NEID and PEID is used to uniquely identify a single Processing Element within a cluster.

physical addressing-A method of station addressing using a unique identifier such that at most one station responds to the specified address.

PIMA-Portable Intelligent Maintenance Aid-A system resembling a laptop computer which will initiate the maintenance built in testing (M-BIT), interrogate AFTA for fault information logged during a mission, and extract maintenance records for system components.

PMD-Physical layer Medium Dependent-The standard which defines the physical medium that is used for the data communications channel on a network.

presence test-The polling of various components to determine if each is active and synchronized. The testing may be performed on members of virtual groups or on the virtual groups themselves.

primitive-A function or procedure that one entity provides to another. The primitive definition specifies the inputs, outputs, and data formats for the primitive.

PROM-Programmable Read Only Memory-A form of computer memory that will store a permanent copy of one or more subroutines specifically intended for use by a particular microprocessor. PROM's allow for a certain level of hard-wired software control over the processor.

quadruplex-A virtual group consisting of four processing sites.

rate group dispatcher-An RG4 task that is responsible for controlling the execution of the rate group tasks and providing reliable communication between the rate group tasks throughout the system.

Register Transfer Level (RTL) VHDL-A behavioral format which specifies the functionality of a block from the standpoint of random combinational logic and/or synchronous registers. For the purpose of the AFTA NE development, RTL is defined to be synthesizeable behavioral VHDL, that is, a behavioral VHDL description that is suitable for input to a synthesis tool.

reprocurement-The act of obtaining new parts to replace parts in an existing system, or to build additional copies of an existing design.

RG-Rate Group-A set of tasks whose iteration rate is well-defined and whose execution times do not exceed the iteration frame (the inverse of the iteration rate).

RISC-Reduced Instruction Set Computer-A type of microprocessor which utilizes a limited set of machine language instructions to allow for more rapid execution of those instructions and thus greater throughput for the computer.

RTS-Run Time System

SAVA-Standard Army Vetronics Architecture

sequential I/O-Input/Output processes that require the managing virtual group to completely supervise the activity. In other words, the virtual group must block itself until the I/O is finished.

SERP-System Exchange Request Pattern-A string of bytes describing the current state of the input and output buffers for each processor in the system. The SERP is used to determine if packets can be sent from one virtual group to another. The LERP from each FCR is exchanged using a source congruency to generate the SERP. Because the SERP originates from a source congruency exchange, it can be considered congruent throughout all functioning FCRs.

SIFT-Software Implemented Fault Tolerance-System fault tolerance functions achieved primarily through operating system programming rather than primarily through dedicated hardware.

simplex-A virtual group consisting of only one processing site.

single-source data-An element of information which originates from a single point. Examples of single-source data include sensor readings, input values, and syndromes. Single-source data must be distributed to fault-masking groups using a source congruency exchange to maintain Byzantine resilience.

sortie availability-One minus the probability that the vehicle is prevented by the AFTA from beginning a mission at the desired time.

source congruency-A type of exchange used to distribute data from a single source, such as an input device, to members of a fault-masking group. The source congruency, which is also known as a class 2, 2-round exchange, or interactive consistency, is a primary requirement for a Byzantine resilient system.

station-A device connected to a network that can transmit or receive data over the network. Often a station is a processing site. In the FTDB, a station can be a redundant processing site.

structural VHDL-A level of description that specifies a VHDL architecture by defining interconnections of instantiations of VHDL entities . A structural description resembles a conventional netlist.

syndrome-A bit field indicating the observance of unusual behavior somewhere in the system. Syndromes can be used in an attempt to diagnose and repair faults in the system.

System FDI- A process that will coordinate system status and fault information as well as testing and analyzing shared components.

task migration-The movement of a necessary task from a failed processor to another processor within the same fault containment region.

test bench-A model of a test fixture that is used to test a device being designed with VHDL. The test bench is written in VHDL and provides a non-proprietary way of stimulating and monitoring a design in a simulator.

testability-The ability to unambiguously ascertain the functionality of each Line Replaceable Module of the AFTA.

TF/TA/NOE-Terrain Following/Terrain Avoidance/Nap of the Earth-A typical helicopter mission application for which the AFTA will be designed.

THT-Token Holding Timer-A method used with token passing media access protocols to limit the amount of time each station can transmit on the network.

timeout-A value of time used to monitor skew between processors of an FMG. All processors in an FMG should be synchronized to within one timeout value, so if a processor does not respond within the timeout period, that processor is considered faulty, and the other processors will continue uninhibited. Timeouts are necessary on the AFTA to prevent faulty processors from halting the system.

timestamp-A 32-bit quantity that indicates the relative time within the cluster. The Network Element places a timestamp in the input info block for each packet successfully delivered to a virtual group.

TNR-Transient NE Recovery-The procedure by which a Network Element which has suffered a transient fault is reintegrated into the cluster. The first part of TNR is similar to the ISYNC procedure. TNR also specifies the realignment of the Network Element state.

transient recovery policy-A recovery option whereby the faulty component is immediately disabled and an attempt is made to reintegrate the component into the system.

triplex-A virtual group consisting of three processing sites.

validation-The process of demonstrating that an implemented system correctly performs its intended functions under all reasonably anticipated operational scenarios.

validity-In a Byzantine resilient system, a condition in which all functioning members of a fault-masking group are guaranteed to possess correct data. The validity condition also implies the agreement condition.

vehicle reliability-One minus the probability that the vehicle is lost due to failure of the AFTA.

VG-virtual group-A grouping of one or more processors to form a virtual (possibly redundant) single processing site. All processors in a virtual group execute the same instruction stream. If a virtual group has more than one member, those members must reside in different FCRs. Virtual groups of 3 or more members are known as fault-masking groups.

VHDL-VHSIC Hardware Description Language-A language for specifying hardware design. VHDL designs can be expressed in a behavioral or a structural method. VHDL also defines a simulation environment and incorporates an intrinsic sense of time.

VHSIC-Very High Speed Integrated Circuit-A Government-funded project to develop technologies to be applied to new, high speed integrated circuits. The VHSIC Hardware Description Language (VHDL) was developed under the VHSIC program.

VID-Virtual Identifier-The name by which a virtual group is known to the system. Also, sometimes used as a synonym for virtual group.

voted message-A message sent by all members of a redundant processing group. This message type is only used when exact consensus among all redundant members is expected. This is also known as a Class 1 message.

voter test-A test of the Network Element voting mechanism that seeds non-congruent values selectively on each channel of a fault masking group.

WAN-Wide Area Network-A network topology that interconnects computer systems separated by long distances. WAN systems usually use packet switched technology.

watchdog timer-A simple timekeeper that will monitor operations in both the Processing Elements and the Network Elements to keep the hardware and software from wandering into undesirable states.

working group-The set of FCRs in a cluster which are synchronized and in the operational phase. An FCR which suffers a fault drops out of the working group. The working group may attempt to reintegrate the failed FCR into the working group.

WPV-Weight Power Volume-These are physical characteristics used to describe the AFTA.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1992	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE
Advanced Information Processing System: The Army Fault Tolerant Architecture Conceptual Study - Volume I : Army Fault Tolerant Architecture Overview

5. FUNDING NUMBERS
WU 505-64-52-53
C NAS1-18565
TA 14

6. AUTHOR(S)
R. E. Harper, L. S. Alger, C. A. Babikyan, B. P. Butler, S. A. Friend, R. J. Ganska, J. H. Lala, T. K. Masotto, A. J. Meyer, D. P. Morton, G. A. Nagle, and C. E. Sakamaki

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA CR-189632, Volume I

11. SUPPLEMENTARY NOTES
Technical Monitor: Carl R. Elks, Acrostructures Directorate, AVRADA, AVSCOM, Langley Research Center, Hampton, VA

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified- Unlimited

Star Category 62

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Army Avionics Research and Development Activity (AVRADA) is pursuing programs that would enable effective and efficient management of large amounts of situational data that occurs during tactical rotorcraft missions. The "Computer-Aided Low Altitude Night Helicopter Flight Program" has identified automated Terrain Following/Terrain Avoidance, Nap of the Earth (TF/TA, NOE) operation as key enabling technology for advanced tactical rotorcraft to enhance mission survivability and mission effectiveness. The processing of critical information at low altitudes with short reaction times is life-critical and mission critical necessitating a ultrareliable/high throughput computing platform for dependable service for flight control, fusion of sensor data, route planning, near-field/far field navigation, and obstacle avoidance operations.

To address these needs the Army Fault-Tolerant Architecture (AFTA) is being designed and developed. This computer system is based upon the Fault-Tolerant Parallel Processor (FTPP) developed by Charles Stark Draper Laboratory, Inc. (CSDL). AFTA is hard real-time, Byzantine fault-tolerant parallel processor which is programmed in the ADA language.

This document describes the results of conceptual study (Phase I of a 3-year project) of the AFTA development. This document contains detailed descriptions of the program objectives, the TF/TA NOE application requirements, architecture overview, hardware design, operating systems design, analytical models and development plan.

14. SUBJECT TERMS
Fault-tolerant, Real-time digital computer, Terrain-following/terrain avoidance helicopter operation

15. NUMBER OF PAGES
144

16. PRICE CODE
A07

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT